



Evaluierung von REST Frameworks für Android

im Kontext des Revex2020 Projekts

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, 26. Januar 2016

Kurzfassung

Schlüsselwörter

REST, Android, Java

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	4
3	Android	7
3.1	Überblick	7
3.2	Architektur	7
3.3	Cross Compiling	9
3.4	Android und Java	9
4	Rest	10
5	Vergleich	11
5.1	Retrofit	11
5.1.1	Funktionsweise	11
5.2	Spring for Android	13
5.2.1	Funktionsweise	14
5.3	Jersey	15
	Literatur	16
	Wissenschaftliche Literatur	16
	Online Referenzen	17

Abbildungsverzeichnis

1.1	Use-Case-Diagramm	2
2.1	Github Stars, abgerufen am 25.09.2015	5
2.2	Stackoverflow Questions, abgerufen am 24.09.2015	5
2.3	Github Commits, abgerufen am 25.09.2015	5
2.4	GitHub Last Commit, abgerufen am 28.09.2015	6
3.1	Marktanteil von mobilen Betriebssystemen [12]	7
3.2	Android Architektur [26]	8

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 **Stackholdern**, 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 (**IKT**) in den letzten Jahren in den Unternehmen stetig zugenommen[24].

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 SOAP¹ oder REST² umgesetzt [4].

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**. 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 ~~die Evaluierung von~~ REST-Frameworks für Android ist noch relativ neu, ~~deswegen existieren noch nicht viele Publikationen~~, um 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[10]. 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.

¹ Simple Object Access Protocol

² Representational State Transfer

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 [15].

Revex2020 ist ein Forschungsprojekt zur Revitalisierung von Wasserkraftwerken, das in Kooperation mit dem Institut für Energietechnik und Thermodynamik entwickelt wird [8]. 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.

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.

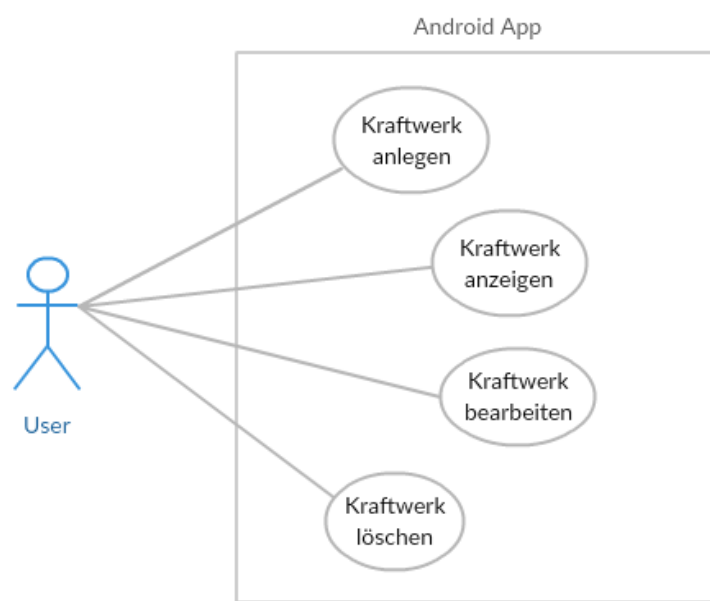


Abbildung 1.1: Use-Case-Diagramm

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"[10] 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 [25] weggelassen, da es für die clientseitige Verarbeitung irrelevant ist, welche Ressourcentypen serverseitig implementiert werden können.

Allgemein:

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

Implementierung mit REST-Framework:

- Wie lange wird benötigt um das Framework einzubinden? (Zeitdauer) 
- Welche HTTP-Verben 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.)
- Wie erfolgt die Identifikation einzelner Ressourcen? (Aufruf der URL)
- Wird das **HATEOAS** Konzept unterstützt?

Erweiterte Technische Fähigkeiten

- Definiert das Framework eine eigene **IDL³**?
- Wie wird ~~der Bereich~~ Sicherheit gehandhabt? (Authentifizierung, Verschlüsselung)
- Werden andere Protokolle ~~außer HTTP noch~~ unterstützt?
- Gibt es eine Möglichkeit für asynchronen Nachrichtenaustausch?
- Wird transaktionales Verhalten vom Framework unterstützt? (**ACID-Eigenschaften**)

³ Schnittstellenbeschreibungssprachen

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://restitute.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. ~~Es wurde daher~~ 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 zur Verfügung steht. Eine Studie von Chris Parnin[5] 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"[1] 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/>

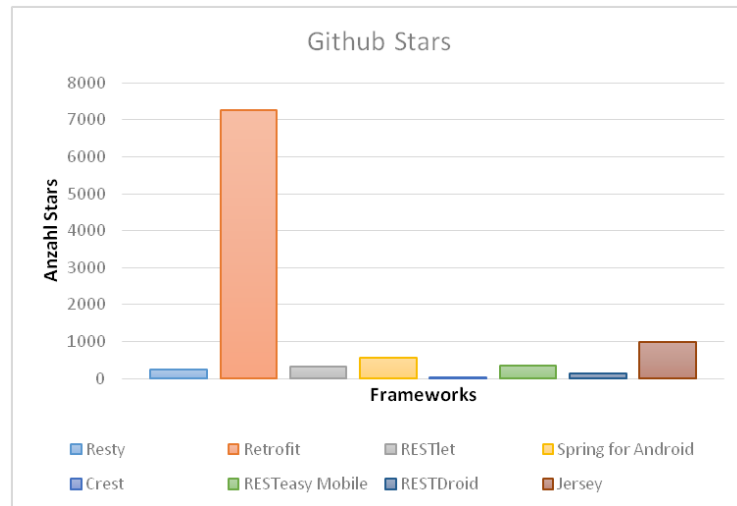


Abbildung 2.1: Github Stars, abgerufen am 25.09.2015

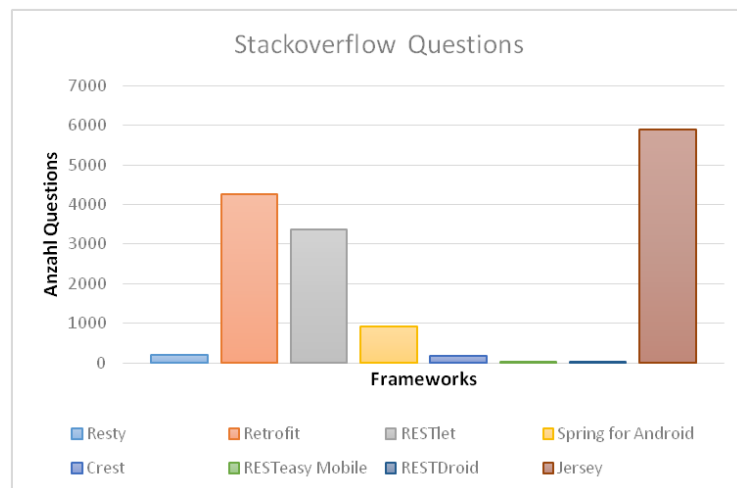


Abbildung 2.2: Stackoverflow Questions, abgerufen am 24.09.2015

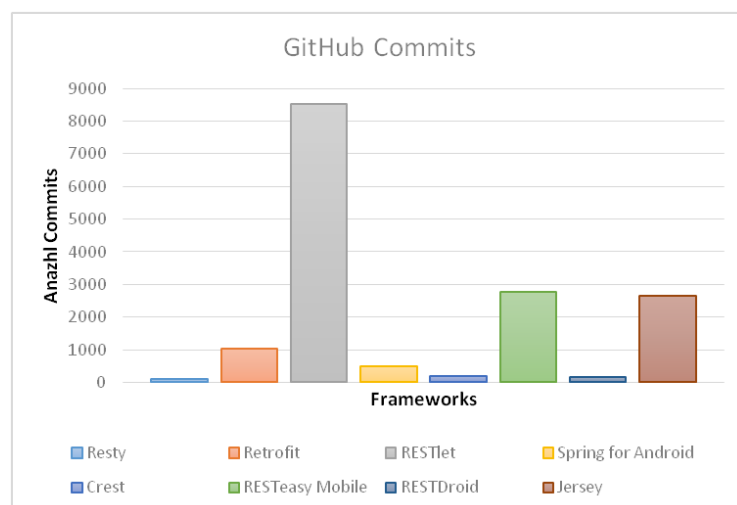


Abbildung 2.3: Github Commits, abgerufen am 25.09.2015

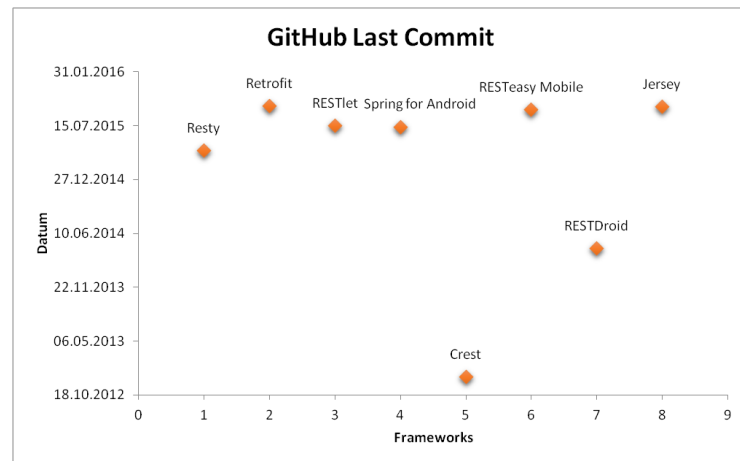


Abbildung 2.4: GitHub Last Commit, abgerufen am 28.09.2015

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

- Retrofit
- Jersey
- Spring for Android

3 Android

3.1 Überblick

Android ist ein Betriebssystem, welches primär für Smartphones und Tablets konzipiert ist. Das Betriebssystem basiert auf einem Linux Kernel und wird von der Open Handset Alliance (gegründet von Google) entwickelt [18]. Android ist eine freie Software und das am schnellsten wachsende mobile Betriebssystem. Der Marktanteil von Android ist seit 2014 über 80% und soll sich auch in den darauffolgenden Jahren bei dieser Prozentzahl halten, wie in der Abbildung 3.1 zu sehen ist.

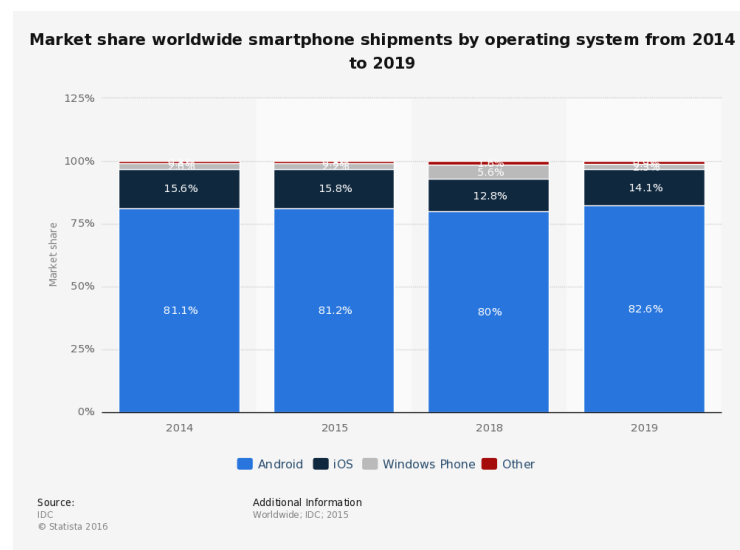


Abbildung 3.1: Marktanteil von mobilen Betriebssystemen [12]

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 ~~Annehmerkreis~~ **Annehmerkreis** 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 [26].

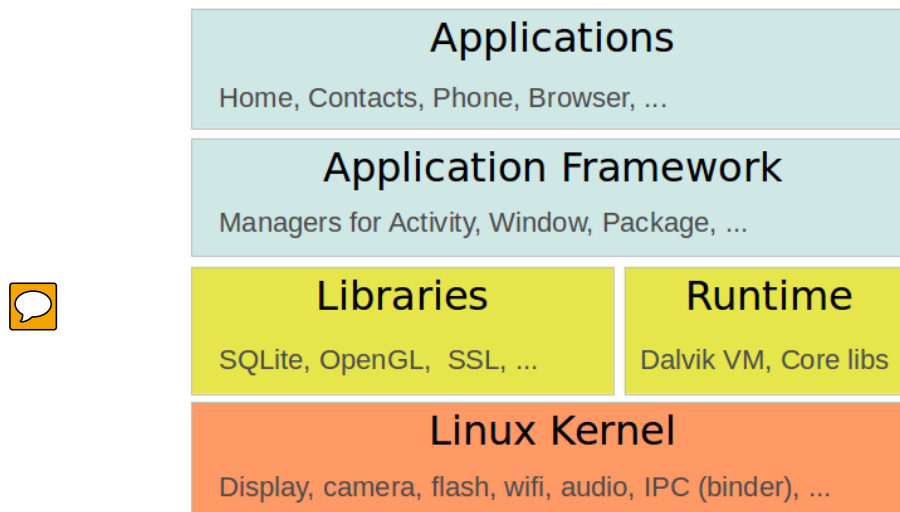


Abbildung 3.2: Android Architektur [26]

Beschreibung der Software Komponenten der Android Architektur [18], [26]:

- **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.

- **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 und den Java Kernbibliotheken. Die Dalvik Virtual Machine 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, mit 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 eine App ausgeführt werden kann [7].

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 [2]. Dabei unterstützt die Android Java API einen Großteil der packages welche in der Java Standard Edition (SE) Bibliothek vorhanden sind. Einige packages wurden aber weggelassen, da sie auf einer mobilen Plattform keinen Sinn machen [17], wie etwa das Drucken (javax.print). Jedoch wurden zusätzlich einige Drittanbieter Bibliotheken hinzugefügt, um Entwicklern die Arbeit zu erleichtern, beispielsweise die Apache HttpComponents Bibliothek (org.apache.commons.httpclient) [3].

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

4 Rest

Der Architekturstil Representational State Transfer, kurz 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 HTTP oder URIs. Er legte nur Kernprinzipien fest, die mit unterschiedlichen Protokollen umgesetzt werden können. Zum Beispiel wie Ressourcen im WEB identifiziert oder adressiert werden [9].

Es werden dabei folgende 5 Kernprinzipien unterschieden [23]:

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*.

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



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 Vergleich

5.1 Retrofit

Retrofit ist ein typischerer HTTP Client für Android und Java, welcher von Square Open Source entwickelt wurde [22]. Das Framework baut auf OkHttp auf, welches die Kommunikation auf der Netzwerkebene übernimmt [21]. 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 JSON oder 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 [11]. Dadurch wurde die GSON Bibliothek auch im Rahmen der Bachelorarbeit zum Übertragen von Daten verwendet.

5.1.1 Funktionsweise

Um Retrofit verwenden zu können, müssen folgende Abhängigkeiten zum Projekt hinzugefügt werden:

```
1 dependencies {  
2     compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'  
3     compile 'com.squareup.retrofit:converter-gson:2.0.0-beta2'  
4     compile 'com.google.code.gson:gson:2.4'  
5     compile 'com.squareup.okhttp:okhttp:2.4.0'  
6 }
```

Listing 5.1: app\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.

```
1 private static final String BASE_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(BASE_URL)
7     .addConverterFactory(GsonConverterFactory.create())
8     .build();
9
10 return retrofit.create(serviceClass);
11 }

```

Listing 5.2: Retrofit Builder

Die Endpoints werden durch spezielle Annotationen bei 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 Klammer gekennzeichnet wurde, durch den dazugehörenden Parameter ersetzt (siehe Listing 5.3, 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 5.3, Zeile 13).

Der return Wert ist dabei immer ein parametrisiertes `Call<T>` Objekt, beispielsweise `Call<PowerPlant>`. Wir 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. Dadurch kann zum Beispiel vor jeder Anfrage an den Server, ein Token hinzugefügt werden, welches angibt ob ein User eingeloggt ist.

```

1 public static <S> S createServiceWithAuthToken(Class<S> serviceClass
  ↪ , final AuthToken token) {
2     Interceptor interceptor = new Interceptor() {
3         @Override
4         public Response intercept(Chain chain) throws IOException {
5             Request newRequest = chain.request().newBuilder()
6                 .addHeader("Accept", "application/json")
7                 .addHeader("X-Auth-Token", token.getToken())
8                 .build();
9
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

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 [13]. Spring for Android ist ein Projekt von Spring, dass die Entwicklung von Andorid-Apps vereinfachen soll. Es stellt dabei ausgewählte Komponenten von Spring, gebündelt für Android zur Verfügung [19]. Das sind im speziellen

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

Spring for Android unterstützt dabei nicht Dependency Injection, eine der Kernfunktionen des Core Spring Frameworks. Es ist daher nicht möglich ~~Inversion of Control~~ auf Android Plattformen zu verwenden [6].

Grundbaustein des Frameworks ist die Klasse `RestTemplate`, welche Teil des im Jahr 2009 erschienen Frameworks **SSpring for MVCist**. Diese Klasse ermögliche Java-Entwicklern eine High-Level-Abstraktion von untergeordneten Java-APIs, wie HTTP Clients. `RestTemplate` for Android unterstützt dabei auch die gzip Komprimierung und die Übertragung von JSON und XML Daten, indem **POJO** Objekte automatisiert umgewandelt werden [6].

5.2.1 Funktionsweise

Um Spring for Android verwenden zu können, muss folgende Abhängigkeit zum Projekt hinzugefügt werden:

```

1 dependencies {
2     compile 'org.springframework.android:spring-android-rest-template
    ↪ :2.0.0.M1'
3 }

```

Listing 5.5: app\build.gradle

Mithilfe der Klasse `RestTemplate` werden HTTP Anfragen zusammengebaut, um mit der Server REST-Schnittstelle kommunizieren zu können.

```

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

```

Listing 5.6: Anfrage um alle Kraftwerke zu erhalten

Die Klasse `RestTemplate` ist dabei das Kernstück für den clientseitigen Zugriff auf einen RESTful Service. 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 ein Message Converter mit dem JSON Daten verarbeitet werden können.



`RestTemplate` stellt verschiedene Methoden zur Verfügung, um die zugrundeliegenden HTTP Methoden verwenden zu können. Diese sind unter anderem DELETE, GET, HEAD, OPTIONS, POST and PUT [20].^e

5.3 Jersey

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"[14] genauer spezifiziert und ist deshalb ein offizieller Teil von Java. Das Jersey Framework ist dabei eine robuste Open-Source Referenz Implementierung der Spezifikation [16].

Literatur

Wissenschaftliche Literatur

- [2] Ed Burnette. *Hello, Android 4th Edition - Introducing Google's Mobile Development Platform*. 2015. Kap. Appendix 1: Java vs. the Android Language and APIs.
- [4] Jens Bertram und Carsten Kleiner. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Xpert.press. Springer, 2012. Kap. 17: Mobile Apps in Enterprise-Anwendungen unter Berücksichtigung von Sicherheitsaspekten, S. 253–267.
- [5] 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).
- [6] Anthony Dahanne. *Instant Spring for Android Starter*. Packt Publishing, 2013.
- [8] Eduard Doujak. “Ein Beitrag zur technisch ökonomischen Analyse von Verschlußkörpern bei der Beurteilung über Revitalisierungs- bzw. Modernisierungsmaßnahmen von Wasserkraftanlagen”. TU Wien, 2000.
- [9] Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. Diss. 2000.
- [10] 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>.
- [15] Michael Kern. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Xpert.press. Springer, 2012. Kap. 7: Eine neue Generation von Geschäftsanwendungen, S. 95–106.
- [16] 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. ACM, 2011, S. 67–72.
- [17] 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] Rajinder Singh. “An Overview of Android Operating System and Its Security Features”. In: *International Journal of Engineering Research and Applications* (2014).
- [20] SpringSource. *Spring Framework Reference Documentation*. 2015. Kap. 27.10. Accessing RESTful services on the Client, S. 661 –667.
- [23] 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.
- [24] Claudia Linnhoff-Popien und Stephan Verclas. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Xpert.press. Springer, 2012. Kap. 1: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse, S. 3–14.

Online Referenzen

- [1] 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/>.
- [3] Ed Burnette. *Java vs. Android APIs*. Letzter Besuch: 26.01.2016. URL: <http://www.zdnet.com/article/java-vs-android-apis/>.
- [7] Hans-Christian Dirscherl. *Unterschiede zwischen JVM und Dalvik VM*. Letzter Besuch: 26.01.2016. URL: <http://www.pcwelt.de/ratgeber/Unterschiede-zwischen-JVM-und-Dalvik-VM-Smartphone-Grundlagen-1005308.html>.
- [11] Roger Hu. *Consuming APIs with Retrofit*. Letzter Besuch: 31.10.2015. URL: <https://guides.codepath.com/android/Consuming-APIs-with-Retrofit>.
- [12] IDC. *Market share worldwide smartphone shipments by operating system from 2014 to 2019*. Letzter Besuch: 04.01.2016. URL: <http://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems>.
- [13] ITWissen.info. *Spring*. Letzter Besuch: 02.01.2016. URL: <http://www.itwissen.info/definition/lexikon/Spring-Spring-framework.html>.
- [14] *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>.
- [19] *Spring for Android*. Letzter Besuch: 02.01.2016. URL: <https://github.com/spring-projects/spring-android>.
- [21] Square. *OkHttp*. Letzter Besuch: 31.10.2015. URL: <http://square.github.io/okhttp/>.
- [22] Square. *Retrofit*. Letzter Besuch: 19.10.2015. URL: <http://square.github.io/retrofit/>.
- [25] Martina Szybiak. *Kurz & Gut: Ressourcen*. Letzter Besuch: 31.10.2015. URL: <http://www.se.uni-hannover.de/pub/File/kurz-und-gut/ws2011-labor-restlab/RESTLab-Ressourcen-Martina-Szybiak-kurz-und-gut.pdf>.
- [26] 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>.