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, 19. Februar 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 **5**

3 Android **8**
- 3.1 Überblick . . . 8
- 3.2 Architektur . . . 8
- 3.3 Cross Compiling . . . 10
- 3.4 Android und Java . . . 10

4 Rest **11**

5 Vorstellung der Frameworks **12**
- 5.1 Retrofit . . . 12
  - 5.1.1 Allgemein . . . 12
  - 5.1.2 Funktionsweise . . . 13
- 5.2 Spring for Android . . . 15
  - 5.2.1 Allgemein . . . 15
  - 5.2.2 Funktionsweise . . . 15
- 5.3 Jersey . . . 18
  - 5.3.1 Allgemein . . . 18
  - 5.3.2 Workaround für Android . . . 18
  - 5.3.3 Funktionsweise . . . 19
- 5.4 AndroidAnnotations . . . 21
  - 5.4.1 Allgemein . . . 21
  - 5.4.2 Funktionsweise . . . 22

6 Vergleich **24**
- 6.1 Gegenüberstellung der Frameworks . . . 24
- 6.2 Relevante Aspekte für Revex2020 . . . 31
- 6.3 Persönliches Fazit zu den Frameworks . . . 31

7 Zusammenfassung **32**

Literatur **33**
- Wissenschaftliche Literatur . . . 33
- Online Referenzen . . . 34

Abbildungsverzeichnis

1.1 Use-Case-Diagramm 2

2.1 Github Stars, abgerufen am 25.09.2015 6
2.2 Stackoverflow Questions, abgerufen am 24.09.2015 6
2.3 Github Commits, abgerufen am 25.09.2015 6
2.4 GitHub Last Commit, abgerufen am 28.09.2015 7

3.1 Marktanteil von mobilen Betriebssystemen [10] 8
3.2 Android Architektur [11] 9

6.1 Zeitmessung der GET Requests 26
6.2 Zeitmessung der POST Requests 27
6.3 CPU Auslastung 27
6.4 benötigter RAM 28

Liste der Listings

5.1 Retrofit Builder 13
5.2 Auszug aus dem PowerPlantService 13
5.3 Hinzufügen des Tokens, für gültigen Login 14
5.4 GET Request um alle Kraftwerke zu erhalten 16
5.5 Abfrage der Daten zu einem speziellen Kraftwerke 16
5.6 POST Request um ein neues Kraftwerke anzulegen 17
5.7 Workaround Jersey Client auf Android 19
5.8 GET Request 19
5.9 Auszug aus dem PowerPlantService Interface 23
5.10 Interceptor für das setzen eines zusätzlichen HTTP Header Feldes 23

Abkürzungen

CPU Central Processing Unit oder Prozessor

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

POJO Plain Old Java Object

REST Representational State Transfer

SOAP Simple Object Access Protocol

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 Endgräte 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 die 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.

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.

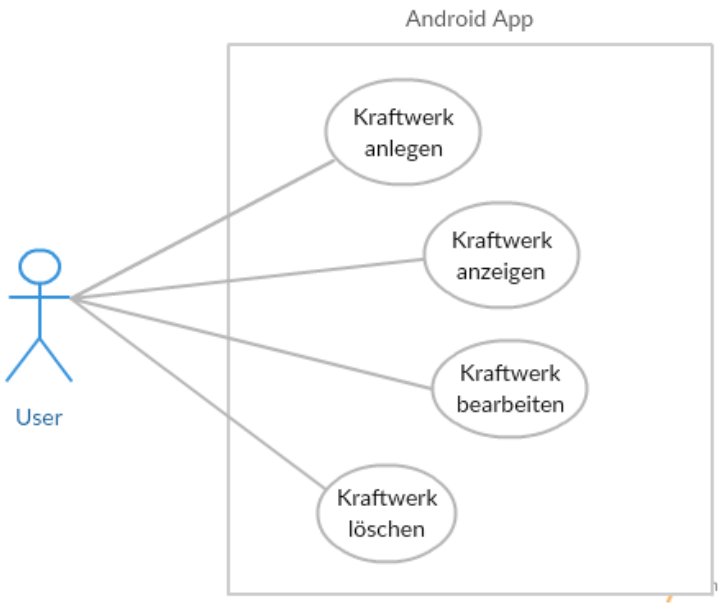


Abbildung 1.1: Use-Case-Diagramm

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 [6] 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 Hypertext Transfer Protocol (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 Resourcen 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 benötige 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 der Bereich Sicherheit gehandhabt? (Authentifizierung)
- Werden andere Protokolle außer HTTP noch unterstützt?
- Wird transaktionales Verhalten vom Framework unterstützt? (ACID-Eigenschaften)
- Unterstützt das Framework die Entwicklung von Server Applikationen?
- Bietet das Framework zusätzliche Dienst fernab der REST-Kommunikation an?

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. 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[7] 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 Overflow[1] und der Stars auf GitHub[2] wurden Rückschlüsse auf die Popularität der einzelnen Frameworks gezogen.

In dem Artikel “How to identify a strong open source project"[8] 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/

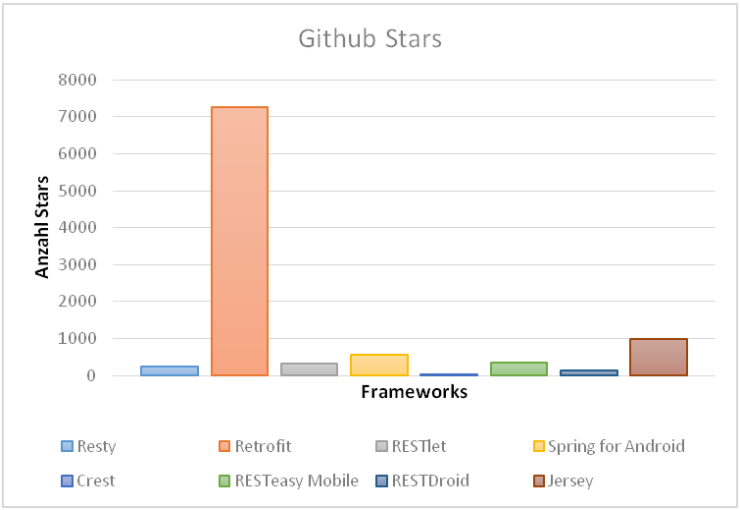


Abbildung 2.1: Github Stars, abgerufen am 25.09.2015

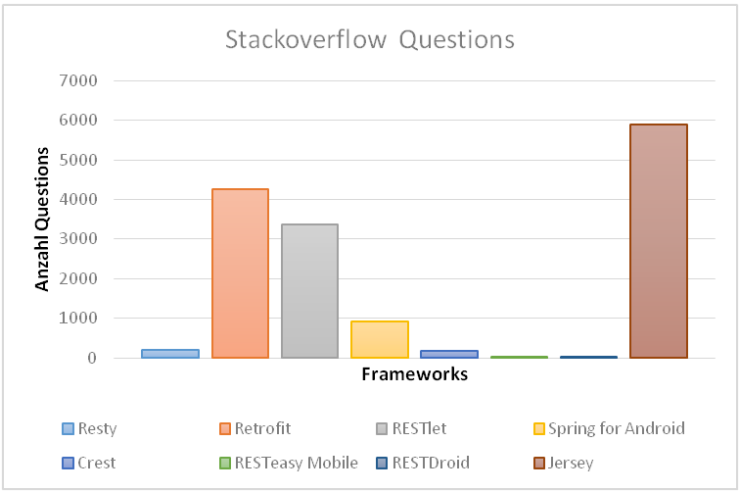


Abbildung 2.2: Stackoverflow Questions, abgerufen am 24.09.2015

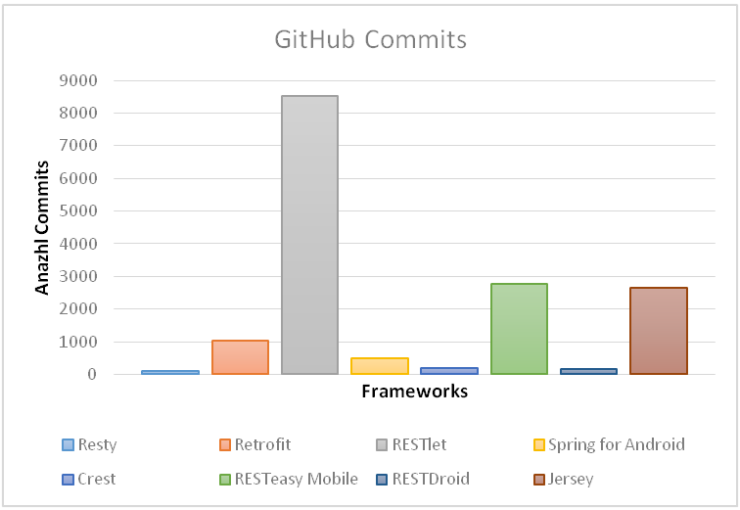


Abbildung 2.3: Github Commits, abgerufen am 25.09.2015

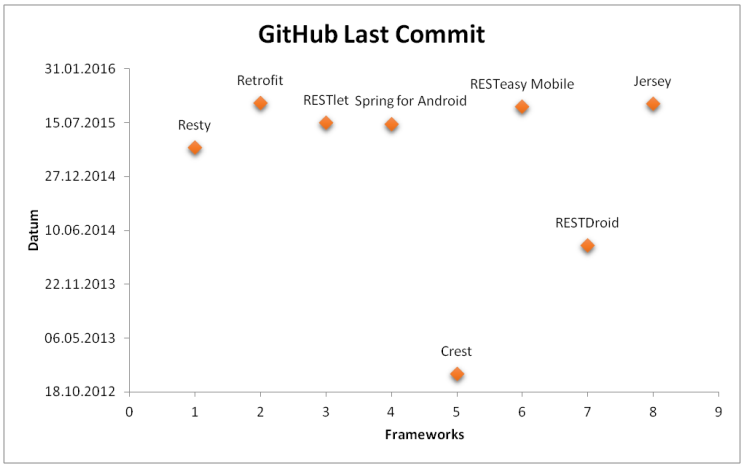


Abbildung 2.4: GitHub Last Commit, abgerufen am 28.09.2015

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

- Retrofit
- Jersey
- Spring for Android

3 Android

3.1 Überblick

Android ist ein Betriebssystems, welches primär für Smartphones und Tablets konzeptioniert ist. Das Betriebssystem basiert auf einen Linux Kernel und wird von der Open Handset Alliance (gegründet von Google) entwickelt [9]. 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.

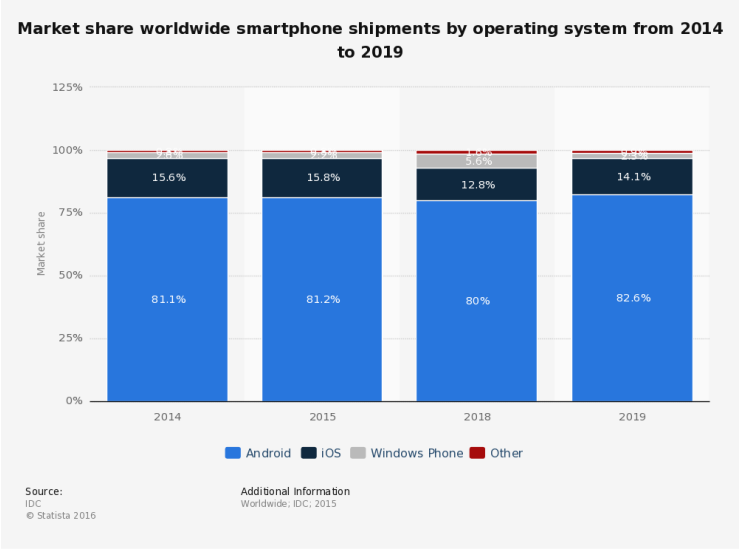


Abbildung 3.1: Marktanteil von mobilen Betriebssystemen [10]

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 [10]. Auf Grund dessen wurde die Evaluierung der REST Frameworks auf Android ausgerichtet, um auch in den folgenden Jahren einen hohen Abnehmerkreis 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 [11].

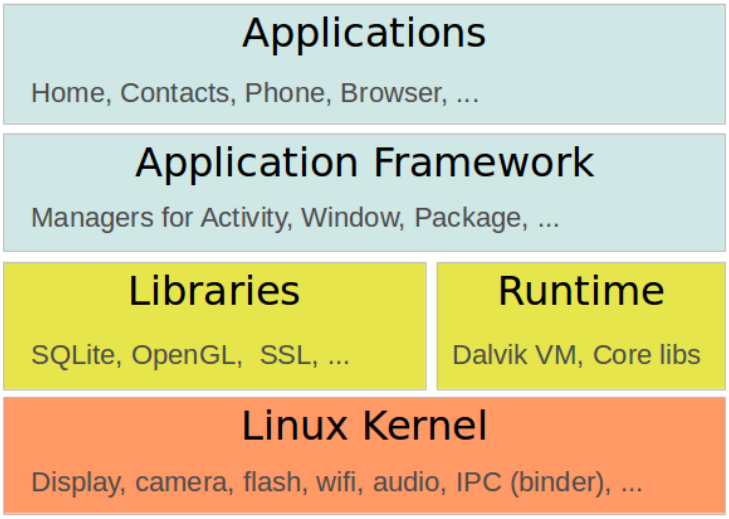


Abbildung 3.2: Android Architektur [11]

Beschreibung der Software Komponenten der Android Architektur [9], [11]:

- **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 habe. Wichtige Dienste sind dabei der Activity Manager, der Resource Manager oder der Content Manager.

- **Native Bibliotheken**
  Die Nativen 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-Datein 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 [12].

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 [13]. 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 [14], 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) [15].

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 [16].

Es werden dabei folgende 5 Kernprinzipien unterschieden [17]:

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 mögliche 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 Dartstellung von Ressourcen
   - POST, für das Erstellen einer Ressource
   - PUT, für das Aktualisieren einer Ressource
   - DELETE, für das Löschen einer Ressource
   - HEAD, um Methadaten 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 wir der Zustand des Clients nicht gespeichert. Der aktuelle Zustand muss vollständig auf Seiten des Clients abgespeichert werden und bei Reqeuests müssen die nötigen Informationen an den Server übermittelt werden.

5 Vorstellung der Frameworks

5.1 Retrofit

5.1.1 Allgemein

Retrofit ist ein typsicherer HTTP Client für Android und Java, welcher von Square Open Source entwickelt wurde [18]. Das Framework baut auf OkHttp auf, welches die Kommunikation auf der Netzwerkebene übernimmt [19]. Retrofit sagt über sich selbst:

> *"Retrofit turns your HTTP API into a Java interface."*, [18, 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 [18]. 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 verwendetste Bibliothek zum Parsen von JSON-Daten darstellt [20]. Dadurch wurde die GSON Biliothek auch im Rahmen der Bachelorarbeit zum Übertragen von Daten verwendet.

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 Klammer gekennzeichnet wurde, durch den dazugehörenden Parameter ersetzt (siehe Listing 5.2, 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 5.2, Zeile 13).

5.1.2 Funktionsweise

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

```
dependencies {
  compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'
  compile 'com.squareup.retrofit:converter-gson:2.0.0-beta2'
  compile 'com.google.code.gson:gson:2.4'
  compile 'com.squareup.okhttp:okhttp:2.4.0'
}
```

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 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 5.1: Retrofit Builder

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

```
public interface PowerPlantService {

  @GET("powerplants")
  public Call<List<PowerPlant>> getPowerPlants();

  @POST("powerplants")
  public Call<PowerPlant> createPowerPlant(@Body PowerPlant
  ↪ powerPlant);

  @DELETE("powerplants/{id}")
  public Call<PowerPlant> deletePowerPlantById(@Path("id") int id);

  @PUT("powerplants/{id}")
  public Call<PowerPlant> updatePowerPlant(@Path("id") int id, @Body
  ↪  PowerPlant powerPlant);
}
```

Listing 5.2: 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 5.3: 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 [21]. 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 [22]. 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. Es ist daher nicht möglich eine lose Koppelung zwischen einzelnen Modulen zu haben [23].

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 JavaScript Object Notation (JSON) und Extensible Markup Language (XML) Daten, indem Plain Old Java Object (POJO) Objekte automatisiert konvertiert werden [23].

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. Diese sind unter anderem `DELETE`, `GET`, `HEAD`, `OPTIONS`, `POST` und `PUT` [24].

5.2.2 Funktionsweise

Um Spring for Android verwenden zu können, muss folgende Abhängigkeit in die `build.gradle` Datei des Projektes eingefügt werden:

```
dependencies {
  compile 'org.springframework.android:spring-android-rest-template:2.0.0.M1'
}
```

Wie bereits oben beschrieben wird die Klasse `RestTemplate` dazu verwendet um HTTP Anfragen zusammenzubauen, um mit der Server REST-Schnittstelle kommunizieren zu können. Es folgt ein kurzes Beispiel eines GET Requests, um vom Server alle vorhanden 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 5.4: GET Request um alle Kraftwerke zu erhalten

Wie man anhand des Beispieles 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 an den Server, das X-Auth-Token hinzuzufügen, welches angibt das ein User eingeloggt ist.

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 5.5, Zeile 3) spezifiziert.

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

Listing 5.5: Abfrage der Daten zu einem speziellen Kraftwerke

Ein POST Request mit Spring for Android sieht folgendermaßen aus:

```
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 5.6: 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"[25] genauer spezifiziert und ist deshalb ein offizieller Teil von Java. Das Jersey Framework ist dabei eine robuste Open-Source Referenz Implementierung der Spezifikation [26].

Die JAX-RS-Client-API Implementierung von Jersey kann mit eine beliebige Web-Service, der auf das HTTP-Protokoll aufsetzt, kommunizieren. Der serverseitige Endpoint muss nicht unbedingt mit JAX-RS implementiert sein.

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 [27].

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 [27].

Die Client API von Jersey unterstützt alle gängigen HTTP Methoden, welche `GET`, `POST`, `PUT`, `DELETE`, `HEAD` und `OPTION` sind [28].

5.3.2 Workaround für 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 [29]. 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.

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

Dennoch ist es noch immer nicht möglich, den Jersey Client ohne Fehler 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 folgende Workaround hilft die

geworfenen Fehler durch fehlenden Abhängigkeiten zu unterdrücken und dennoch alle Funktionen des Jersey Clients in der Android Umgebung zu unterstützen [30].

```
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 5.7: 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 durch folgenden Implementierung ein einfacher `GET` Request realisiert werden, der ein `PowerPlant` Objekt zurückgibt. 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 5.8: GET Request

Im folgenden Beispiel 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, an AndroidAnnotations zu arbeiten [31].

Das Projekt AndroidAnnotations soll dabei helfen, Codefragemente die sich bei der Android-App Entwicklung wiederholen zu reduzieren. Dadurch verringert sich der zu schreibende und wartende Code, wodurch die Entwicklung beschleunigt und die Wartbarkeit verbessert werden kann. Einer der größten Vorteile des Projektes ist die Unterstützung von Dependency Injection. 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 [32].

Die Haupteigenschaften des Projektes sind [33]:

- **Dependency Injection**
  Injection von Views, System Services oder Ressourcen.
- **einfaches Threading-Modell**
  Annotation über Methoden, die angeben ob sie vom UI-Thread oder von einem Hintergrundthread ausgeführen werden sollen.
- **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.

Bei der Kompilierung des Codes werden die verwendeten Annotations 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 [32].

Die Rest API von AndroidAnnotations ist dabei ein Wrapper rund um das Spring for Android `RestTemplate` Modul. Die `@Rest`-Annotation über einem Interfaces ist dabei der Grundbaustein, welche die Schnittstelle genauer spezifizieren. Die Endpoint URL vom Server wird über den `rootUrl` Parameter bei der `@Rest`-Annotation angegeben. Des weiteren muss bei der `@Rest`-Annotation mindestens ein Message Konverter 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 MIME-Typ des Requests oder Response verarbeitet werden kann. Es können auch Interceptoren hinzugefügt werden, welche beispielsweise jeden Request protokollieren oder benutzerdefinierte Authentifizierung handhaben [34].

Weitere wichtige Annotationen für die Rest Kommunikation sind [34]:

- `@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 is nur ein `@Body` Parameter pro Request zulässig.

5.4.2 Funktionsweise

Um AndroidAnnotations verwenden zu können muss folgendes in die `build.gradle` Datei eingefügt werden:

```
apply plugin: 'com.android.application'
apply plugin: 'com.neenbedankt.android-apt'

dependencies {
  apt "org.androidannotations:androidannotations:4.0-SNAPSHOT"
  compile "org.androidannotations:androidannotations-api:4.0-SNAPSHOT"
  compile "org.androidannotations:rest-spring:4.0-SNAPSHOT"
  compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
  compile 'com.fasterxml.jackson.core:jackson-core:2.6.0'
  compile 'com.fasterxml.jackson.core:jackson-databind:2.6.0'
}

apt {
  arguments {
    androidManifestFile variant.outputs[0].processResources.
    ↪ manifestFile
  }
}
```

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 [35].

Die beispielhafte Implementierung eines AndroidAnnotation Rest Client kann man aus folgendem Listing 5.10 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` verwendet.

```
1  @Rest(rootUrl = "https://revex.inso.tuwien.ac.at/api",
2      converters = { MappingJackson2HttpMessageConverter.class },
3      interceptors = { HttpBasicAuthenticatorInterceptor.class })
4  public interface PowerPlantService {
5
6    @Get("/powerplants")
7    List<PowerPlant> getPowerPlants();
8
9    @Post("/powerplants")
10   public PowerPlant createPowerPlant(@Body PowerPlant powerPlant);
11
12   @Delete("/powerplants/{id}")
13   PowerPlant deletePowerPlantById(@Path int id);
14
15   @Put("/powerplants/{id}")
16   PowerPlant updatePowerPlant(@Path int id, @Body PowerPlant
     ↪ powerPlant);
17 }
```

Listing 5.9: Auszug aus dem PowerPlantService Interface

Beim Listing 5.10 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 5.10: Interceptor für das setzen eines zusätzlichen HTTP Header Feldes

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 gegeben.

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 [36]. 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 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 [37]. 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 [38]. 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 das die entwickelte Software Sicherheitslücken hat oder mehr Zeit für eine korrekte Implementierung benötigt wird [39]. Für alle Frameworks ist eine Dokumentation vorhanden. Die Dokumentation von Retrofit deckt dabei aber nur die grundlegend Funktionsweise ab und ist im Vergleich zu den anderen schwächer. Ein Problem hierbei ist, dass gerade ein großes Update des Frameworks durchgeführt wurde und

die vorhandene Dokumentation dadurch teilweise veraltet ist. 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 gefunden 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 benützen. 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 unterstützten, 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 Dateien 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 [40]. 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 Aktion 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 größteil der User verständlicher [41].

Performance

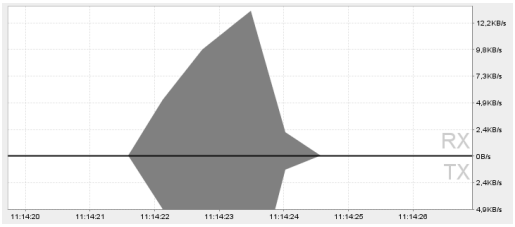
Mobile Geräte haben längst Einzug in das Alltagsleben erhalten, viele Nutzer erweitern die Funktionalität der Geräte mit zahlreichen Anwendungen. Jede installierte App auf einen mobilen Endgerät hat Einfluss auf den Energieverbrauch. Je mehr Anwendungen auf eine Endgerät installiert sind und je öfter diese genutzt werden, desto höher ist der Energieverbrauch und die Betriebs-

zeit sinkt dadurch massiv [42]. Daher ist bei der Entwicklung darauf zu achten, das der Energieverbrauch so gering wie möglich gehalten wird. Auswirkungen auf den Energieverbrauch eines Smartphones haben sowohl der Bildschirm, die Central Processing Unit oder Prozessor (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 [43].

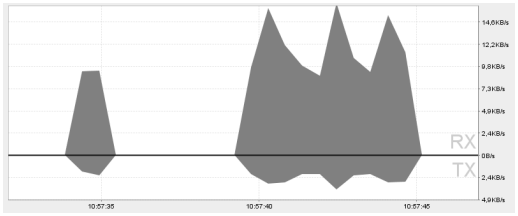
Umso länger und mehr Daten bei einem Request über das Mobilfunknetz oder das WLAN-Modul übertragen werden, desto mehr Energie wird verbraucht. Es sollte daher darauf geachtet werden, dass die Requestdauer möglichst gering ist und nicht unnötige Daten übertragen werden. Des weiteren ist die Rechenleistung von Smartphones wesentlich geringer als jene von Notebooks, um die Betriebszeit zu erhöhen. Daher ist bei der Entwicklung auch darauf zu achten, dass die CPU Beanspruchung der Apps gering gehalten wird, damit das System performant weiterarbeiten kann [44]. 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 [45].

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, wurden jede App einzeln in einem Emulator gestartet und die gleich Abfolge an Funktionsaufrufen durchgeführt.

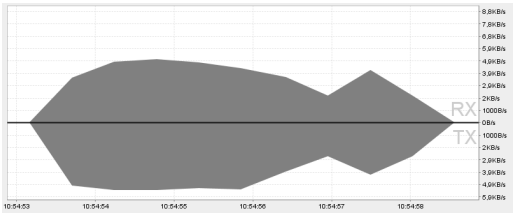
Die Zeitdauer der GET-Requests bei der Abbildung 6.1 setzt sich aus 11 einzeln abgesetzte GET-Requests zusammen. Diese Requests werden benötigt, um die Details zu einem bestimmten Kraftwerke anzuzeigen. Es werden in der Praxis oft mehrere GET-Request hintereinander abgesetzt, daher wurde die Dauer ermittelt 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 ist.



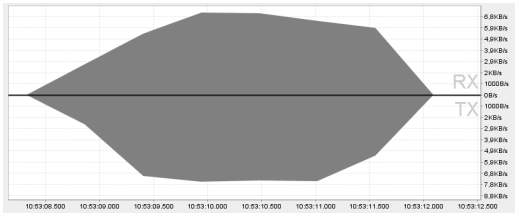
(a) GET Request Retrofit



(b) GET Request Jersey



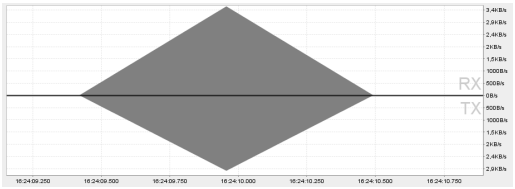
(c) GET Request Spring for Android



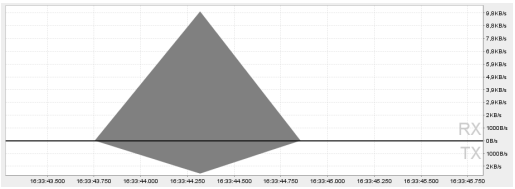
(d) GET Request AndroidAnnotations

Abbildung 6.1: Zeitmessung der GET Requests

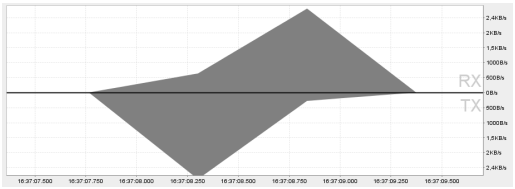
Die Messung der Zeitdauer für einen POST Request ist aus der Abbildung 6.2 zu entnehmen. Dabei wurde die Zeit gemessen bis neu eingegeben Daten, um ein Kraftwerk anzulegen erfolgreich zum Server übermittelt wurden. Es konnte dabei festgestellt werden, dass alle Frameworks circa gleich lange benötigen, 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 [46].



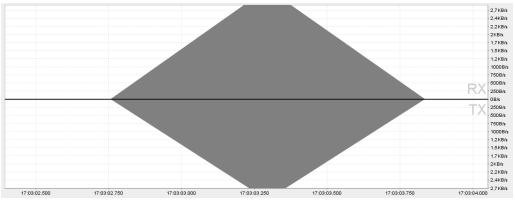
(a) POST Request Retrofit



(b) POST Request Jersey



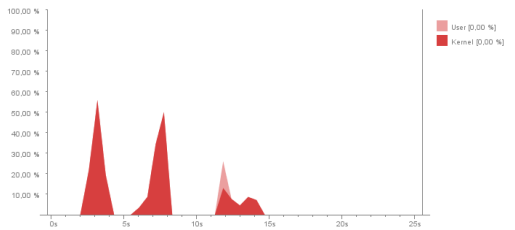
(c) POST Request Spring for Android



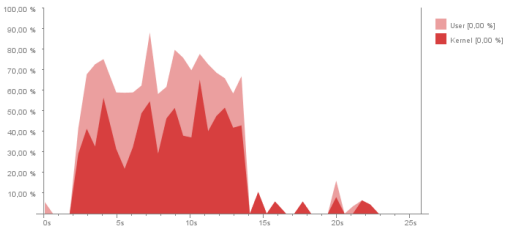
(d) POST Request AndroidAnnotations

Abbildung 6.2: Zeitmessung der POST Requests

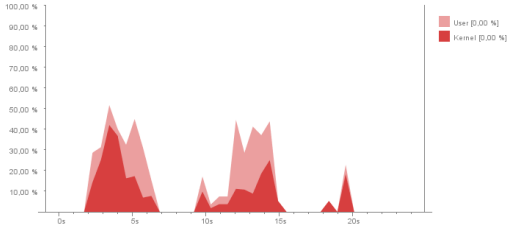
In der Abbildung 6.3 wird die CPU Auslastung der einzelnen Frameworks gegenübergestellt. Aus dieses Abbildung kann entnommen werden, das AndroidAnnotations die CPU am geringsten in Anspruch nimmt und Jersey am stärksten. Bei der Benutzung der Apps ist auch festzustellen, dass AndroidAnnotations und Retrofit am schnellsten arbeiten und eine flüssige Bedienung der Anwendung möglich ist. Jersey ist des öfteren langsam und hat kleine Ruckler währende der Bedienung.



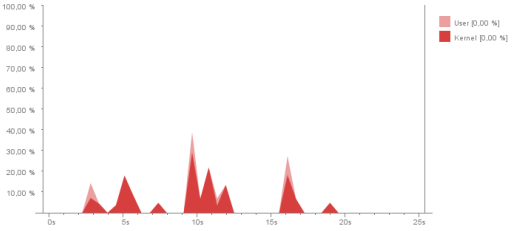
(a) CPU Auslastung Retrofit



(b) CPU Auslastung Jersey



(c) CPU Auslastung Spring for Android

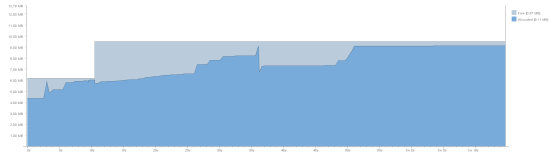


(d) CPU Auslastung Request AndroidAnnotations

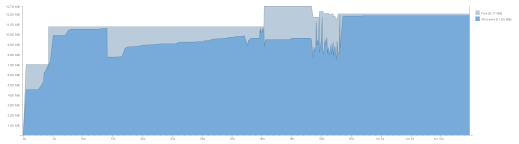
Abbildung 6.3: CPU Auslastung

Speicherplatz

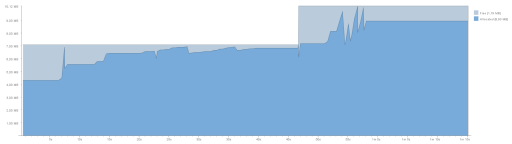
Eine der größten Herausforderungen beim Implementieren von Apps ist der Hardwareunterschied der mobilen Geräte. Die Hardwarekomponenten unterscheiden sich nicht nur in der Displaygröße oder CPU Leistung, sondern auch im Speicherbereich [45]. Sei es nun der zur Verfügung gestellte Random Access Memory (RAM) oder interne Speicher, welcher gelegentlich durch Speicherkarten erweitert werden kann. Je mehr und je größere Apps installiert werden, desto mehr leidet die Performance – insbesondere auf älteren oder Hardware schwachen 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 [47]. Bei der Gegenüberstellung der APK Größe und der maximalen RAM Beanspruchung schnitt AndroidAnnotations am besten ab, Jersey am schlechtesten (siehe Tabelle 6.1).



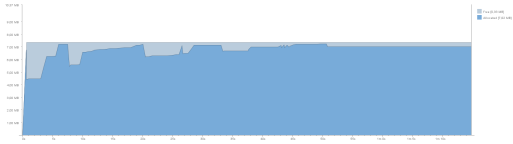
(a) benötigter RAM Retrofit



(b) benötigter RAM Jersey



(c) benötigter RAM Spring for Android



(d) benötigter RAM Request AndroidAnnotations

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 [48]. Dieser Standard wir 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 unterstützten 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 Commited wird oder ein Rollback durchgeführt wird [49].

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

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 Relevante Aspekte für Revex2020

Für die Implementierung einer App sind sowohl Retrofit, Spring for Android und AndroidAnnoations geeignet. Von Jersey als REST Framework ist abzuraten, da dieses Framework nur mithilfe eines Workarounds benützt werden kann. Darüber hinaus schneidet Jersey im Performance- und Speichervergleich am schlechtesten ab. Vergleicht man nun jene anderen drei ist der Unterschied minimal. Retrofit weißt eine etwas schwächere Dokumentation auf, da gerade eine neue Version erschienen ist und die Dokumentation teilweise noch nicht an die neueste Version angepasst wurde. Spring for Android ist im Lesezugriff langsamer, was einen minimalen Nachteil darstellt - da dies das Hauptaugenmerk einer Implementierung für Revex ist. Retrofit und AndroidAnnoations 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. Der größte Vorteil von AndroidAnnoations sind die zusätzlicher Dienste, welche die Implementieren einer Android App erleichtern.

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.

Jersey

Retrofit

Spring for Android

AndroidAnnotations

7 Zusammenfassung

Literatur

Wissenschaftliche Literatur

[1] 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.

[2] 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.

[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*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Xpert.press. 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.

[7] 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).

[9] Rajinder Singh. "An Overview of Android Operating System and Its Security Features". In: *International Journal of Engineering Research and Applications* (2014).

[13] Ed Burnette. *Hello, Android 4th Edition - Introducing Google's Mobile Development Platform*. 2015. Kap. Appendix 1: Java vs. the Android Language and APIs.

[14] 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).

[16] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Diss. 2000.

[17] 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.

[23] Anthony Dahanne. *Instant Spring for Android Starter*. Packt Publishing, 2013.

[24] SpringSource. *Spring Framework Reference Documentation*. 2015. Kap. 27.10. Accessing RESTful services on the Client, S. 661 –667.

[26] 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.

[36] Ursula Widmer und Konrad Bähler. *Open Source Jahrbuch 2006: Zwischen Softwareentwicklung und Gesellschaftsmodell*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Lehmanns Media, 2006. Kap. Open-Source-Lizenzen - Wesentliche Punkte für Nutzer, Entwickler und Vertreiber, S. 165–179.

[37] Alexandra Kees. *Open Source Enterprise Software*. 2015. Kap. 3.5 Communities, S. 30–32.

[38] M.P. Robillard. "What Makes APIs Hard to Learn? Answers from Developers". In: *IEEE Software* 26.6 (2009), S. 27–34.

[39] 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.

[40] Dirk Louis und Peter Müller. *Android: Der schnelle und einfache Einstieg in die Programmierung und Entwicklungsumgebung*.

[41] Jun Gong und Peter Tarasewich. "Guidelines for handheld mobile device interface design". In: *Proceedings of DSI 2004 Annual Meeting*. 2004, S. 3751–3756.

[42] Claas Wilke. "Energieverbrauchsermittlung von Android-Applikationen". In: *Innovationsforum open4INNOVATION2012 regional kooperativ-global innovativ*. Technische Universität Dresden, 2012.

[43] Marcus Vetter. "Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten". Universität Stuttgart, 2012.

[44] 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.

[45] 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.

[46] Marco Glier Dr. Herbert A. Meyer Petra Vogt. "Performance–(k) ein Thema für Usability Professionals?" In: *Usability Professionals* (2005).

[48] 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.

[49] Martin Mensing-Braun. "Transaktionen im Kontext von REST-basierten Web-Diensten". FH Hagenberg, 2015.

Online Referenzen

[6] 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.

[8] 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/.

[10] 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.

[11] 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.

[12] 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.

[15] Ed Burnette. *Java vs. Android APIs*. Letzter Besuch: 26.01.2016. URL: http://www.zdnet.com/article/java-vs-android-apis/.

[18] Square. *Retrofit*. Letzter Besuch: 19.10.2015. URL: http://square.github.io/retrofit/.

[19] Square. *OkHttp*. Letzter Besuch: 31.10.2015. URL: http://square.github.io/okhttp/.

[20] Roger Hu. *Consuming APIs with Retrofit*. Letzter Besuch: 31.10.2015. URL: https://guides.codepath.com/android/Consuming-APIs-with-Retrofit.

[21] ITWissen.info. *Spring*. Letzter Besuch: 02.01.2016. URL: http://www.itwissen.info/definition/lexikon/Spring-Spring-framework.html.

[22] *Spring for Android*. Letzter Besuch: 02.01.2016. URL: https://github.com/spring-projects/spring-android.

[25] *JSR 311: JAX-RS: The JavaTM API for RESTful Web Services*. Letzter Besuch: 03.01.2016. URL: https://jcp.org/en/jsr/detail?id=311.

[27] Doku Jersey API. *Chapter 5. Client API*. Letzter Besuch: 28.01.2016. URL: https://jersey.java.net/documentation/latest/client.html.

[28] 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.

[29] 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.

[30] 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.

[31] *Sponsors*. Letzter Besuch: 09.02.2016. URL: https://github.com/excilys/androidannotations/wiki/Sponsors.

[32] 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.

[33] *Introduction*. Letzter Besuch: 09.02.2016. URL: https://github.com/excilys/androidannotations/wiki.

[34] *Rest API*. Letzter Besuch: 09.02.2016. URL: https://github.com/excilys/androidannotations/wiki/Rest-API.

[35] *Android-apt Overview*. Letzter Besuch: 09.02.2016. URL: https://bitbucket.org/hvisser/android-apt.

[47] Stefanie Schäfers. *Google verdoppelt Größe für Android-APKs*. Letzter Besuch: 19.02.2016. URL: https://entwickler.de/online/mobile/google-verdoppelt-groesse-android-apks-181839.html.