Bachelorarbeit im Studiengang Informatik der Fakultät Elektrotechnik und Informatik

Evaluation und Einsatz von Swift-Frameworks zum Datenaustausch in iOS-Anwendungen

zur Erlangung des akademisches Grades eines Bachelor of Science

angefertigt von

Matthias Strauß

ausgegeben am:	Datum 1
abgegeben am:	Datum 2

Betreuer:

Erstprüfer:	Prof. Dr. Bernhard Glavina
Zweitprüfer:	PrÜfEr EiNtRaGeN
EPOS CAT GmbH:	PrÜfEr EiNtRaGeN

Kinding, DATUM

1 Einleitung

asdfasdf

2 Technologien im Überblick

2.1 Ablauf des Datenaustauschs

Der Datenaustausch hat 3 verschiedene Schritte, die sich abhängig davon, ob Daten empfangen oder gesendet werden, nur in der Reihenfolge ändern.

Falls Daten vom Server aufgerufen werden soll

3 REST APIs

3.1 REST

3.1.1 Herkunft

REST steht für "Represential State Transfer". Es ist der Name eine Architekturstils, der im Jahr 2000 von Roy Thomas Fieldling im Rahmen seiner Doktorarbeit erstmals definiert wurde. Das Ziel war es, die derzeitig aktuellen Architekturstile zu analysieren und dadurch eine Anzahl an Beschränkungen zu ermitteln, die eine funktionale, performante und soziale Anwendung garantieren.

Die Beschränkungen für die Architektur, die Fieldling zusammenstellte, sind wie folgt:

- Client-Server-Stil
- Zustandslosigkeit der Kommunikation
- Zwischenspeicherbare Antworten
- einheitliche Schnittstelle
- hierarchische Ebenen
- Code-on-Demand

Wenn man heute von "RESTful APIsßpricht ist damit oft eine Art der Kommunikation zwischen Klienten und Servern über Hypertext Transfer Protocol (http) gemeint.

3.1.2 Unterschiedliche Implementierungsgrade

Bei der konkreten Entwicklung von APIs, die sich nach REST orientieren, gibt es unterschiedliche Beispiele, wie sehr den Vorgaben der Dissertation von Fieldling gefolgt wurde.

Für die Unterscheidung dieser unterschiedlichen Grade entwickelte Leonard Richardson das "Richardson Maturity Model". Vier Stufen geben an, wie genau es sich um eine "RESTfulÄPI handelt.

- Level 0: Swamp of POX

  Auf dieser Ebene wird http als reines Transportprotokoll verwendet. Oft gibt es nur eine einzige offene Schnittstelle. Die Daten für die Transaktion

werden in eigenen Containern, z. B. XML (POX: Plain Old XML), versendet und verwenden das Protokoll nur als Tunnel.

- Level 1: Resources

  Ebene 1 setzt vorraus, das jeder Ressource ein eigene URI (Unique Ressource Identifier) zugewiesen wird. Für http heißt dies, das jeder Ressource eine URL (z.B. www.example.com/api/user/42) hat.

- Level 2: HTTP Verbs

  Das nächste Level beschränkt die Wahl der Verben der Interaktion auf ihren gedachten Verwendungszweck. Http hat verschiedene Verben zur Verfügung, die alle für eine bestimmte Art der Interaktion gedacht sind. Für die Verwendung in REST haben sich folgende Konventionen ergeben:

  - GET

    GET wird für die Informationsanfrage verwendet. Dabei ändern sich beim Server keine Daten, sie werden nur ausgelesen und weitergegeben.

  - POST

    POST wird verwendet, um neue Daten anzulegen, oder wenn keines der anderen Verben passend ist.

  - PUT

    PUT wird hauptsächlich für die Aktualisierung von Ressourcen verwendet, manchmal aber auch für die Erstellung neuer Ressourcen.

  - DELETE

    DELETE wird, wie der Name besagt, verwendet um Elemente zu entfernen

  Abgesehen von diesen Verben gibt es noch HEAD, OPTIONS, TRACE und CONNECT. Sie sehen aber nur selten oder nie Gebrach.

  Zusätzlich muss der Server auf die Anfragen mit den passenden Statuscodes antworten. Die HTTP Beschreibung hat für viele verschiedene Fälle eigene Codes, die jedoch nach Art der Antwort in verschiedenen Bereichen liegen. Wenn die Anfrage erfolgreich verabeitet werden konnte wird liegt der Code zwischen 200 und 299. So wird bei einer einfachen, erfolgreichen GET Anfrage mit 200(Ok) geantwortet. Bei Problemen auf Clientenseite wird mit einem Code im Bereich 400-499, auf Serverseiten mit 500-599 geantwortet.

- Level 3: Hypermedia Controls

  Ebene drei erfordert den Einsatz von HATEOAS(Hypertext as the engine of Application State). Bei jeder Antwort wird den Ressourcen eine Anzahl von URIs beigefügt, die die Interaktionsmöglichkeiten mit der jeweiligen Ressource beschreiben.Das Ziel ist es, für die gesamte API einen einzigen

Eintrittspunkt zu haben um mit diesem ohne weiterem Vorwissen alle Ressourcen erreichen zu können.

//TODO BEISPIELE / CODE

3.2 Anforderungen an Frameworks

Die Anforderungen an die Frameworks gestalten sich einfach. Zunächst muss eine Anfrage an den Server erstellt werden. Hier gibt es einige Eigenschaften die setzbar sein sollen:

- Pfad

  Wie bereits beschrieben besitzt in REST APIs jede Ressource einen eigenen URI. Daher muss man bei jeder Ressourcenanfrage der jeweilige Pfad mit angegeben werden. Falls sich über den gesamten Anwendungsverlauf die Domain des Servers nicht ändert wäre die Möglichkeit, den Pfad mit dieser zu ergänzen, ein "Pluspunkt"für ein Framework.

- Nutzdaten

  Da nicht nur Daten vom Server geladen, sondern auch Daten zum Server gesendet werden müssen, benötigt man die Möglichkeit, Nutzlast an Nachrichten anzuhängen.

- Content-Type

  (MIME-TYPES)

  Der Content-Type beschreibt, welchen Datentyp die Nutzdaten der Nachricht haben. Das gilt für Anfragen als auch Antworten. Für die vorliegende Verwendung wird der Content-Type stets *CODE* äpplication/json; charset=utf-8"*CODE* sein.

- Accept / Data-Type

  Der Accept-Header beschreibt den Datentyp, den eine Anfrage in der Antwort erwartet. Auch dieser wird in dem Projekt immer äpplication/json; charset=utf-8ßein.

- HTTP Methode

  Je nach gewählter HTTP Methode kann die Antwort des Servers anders ausfallen. Außerdem müssen, wenn REST strikt befolgt wird, die Methoden abhängig von der Intention der Anfrage unterschiedlich gewählt werden (s.o.).

- Weitere HTTP Kopfzeilen

  Es gibt weitere HTTP-Kopfzeilen, die in manchen Kontexten wichtig sein können. Dazu gehören u. A. Cookies, Authentifizierung oder Verfallzeiten. Das gewählte Framework soll die Möglichkeit bieten diese bei Bedarf setzen zu können.

Die Kommunikation mit dem Server soll danach automatisch geschehen. Es müssen jedoch noch zwei Funktionsblöcke übergeben werden können. Einmal für den Fall das der Server die Anfrage erfolgreich verarbeiten konnte und einen der aufgerufen werden soll, wenn Fehler auftreten. Sie sind nötig, um die Daten weiter zu verarbeiten, oder um den Benutzer mitzuteilen, dass es z.B. Probleme mit der Internetverbindung gibt.

4 JSON-Objekt Mapping

//Explizite Type vs Implizierte //Eingebaute JSEONSerialization

Beim Austausch von Daten zwischen Server und Klienten muss mit Objektrepräsentationen gearbeitet werden. Das HTTP erlaubt nur das Versenden von Zeichenketten. Diese Strings müssen alle Informationen beinhalten um das jeweilige Objekt wieder komplett mit allen relevanten Eigenschaften wiederherzustellen. Über die Jahre hinweg haben sich verschiedene Standards entwickelt. Zunächst lag der Fokus auf verschiedene XML Lösungen, doch ab etwa 2006 fand auch JSON immer mehr Verwendung.

4.0.1 Was ist JSON

"JSON (Javascript Object Notation) ist ein einfaches Datenaustausch Format". Ein einfaches Objekt, das die Benutzer einer Website enthält würde wie folgt aussehen:

```
{"users" : [
        { "firstName" : "Max",
          "lastName"  : "Mustermann"
        },
        { "firstName" : "Lisa",
          "lastName"  : "Schmidt"
        }
 ]
}
```

Es basiert auf dem Objektformat von Javascript unter dem ECMA Standard 262 und wird als unter ECMA-404 definiert.

4.0.2 Warum JSON statt XML

Für die Verwendung von JSON im Gegensatz von XML sprechen mehrere Gründe.

- Lesbarkeit

  JSON ist für Menschen und Maschinen einfacher zu lesen.

  //Bilder

  //Bilder mit mehreren kleinen Arrays

- Verfügbarkeit

  Für beide Lösungen gibt es in den meisten Programmiersprachen vorgefertigte Umwandler. Die Ausnahme ist vor allem Javascript, welches nur JSON nativ unterstützt und die String mit *CODE* eval() *CODE* direkt in Objekte umwandeln kann.

- Flexibilität

  JSON benutzt anders als XML kein Schema zur Beschreibung der Daten. Falls sich der Datenaufbau ändert muss kein Schema extra angepasst werden.

4.1 Mapping

Um die Strings die vom Server auf Anfrage versendet werden in Objekte umzuwandeln benötigt man ein vordefiniertes Mapping. Das gleiche gilt für die Vorbereitung zum Versenden von Objekten an den Server. Unter der Voraussetzung das der Aufbau der JSON-Objekte bekannt ist lassen sich bestimmte Key-Value Paare den passenden Objekteigenschaften zuordnen.

//Bild JSON -> UML Objekt

Diese Zuordnung soll möglichst effizient und mit wenig Aufwand erstellbar sein. Meist müssen die Model-Klassen eine Mapping-Klasse erweitern und in ihrem Initialisierer

5 Persistenz

Der dritte und letzte Schritt in der Toolfolge ist die Persistenz. Dieser Begriff beschreibt das dauerhafte Speichern der relevanten Informationen, auch über mehrere Systemstarts hinweg. Hierzu werden in der Regel Datenbanksysteme verwendet. Im Rahmen des iOS-Umfelds ist die Auswahl jedoch geringer als bei anderen Systemen. In diesem Kapitel werden zunächst die wichtigsten Eigenschaften und Herausforderungen beschrieben und die verschiedenen Lösungen für diese untersucht.

5.1 Eigenschaften

Bei der Betrachtung von Persistenzsystemen gibt es verschiedene Eigenschaften, die die Auswahl eingrenzen können. Je nach Anwendungszweck und Projektaufbau sind verschiedene Lösungen besser geeignet als andere.

Erstellung des Modell

Das Modell beschreibt, wie die zu speichernden Datensätze und die Relationen zwischen ihnen aufgebaut sind. Dabei kann es durch die Wahl eines Persistenzsystems zu Einschränkungen in den Datentypen geben. Die Standardeigenschaften, die in der Regel zur Verfügung stehen, sind Zeichenketten, Zahlen und BLOBs (Binary Large Objects). Zu den Relationen gehören

- 1:1 (ein Eintrag wird genau einem oder keinem Eintrag zugeordnet)
- 1:n (ein Eintrag wird genau einem, keinem oder auch mehreren Einträgen zugeordnet)
- n:m (eine beliebige Anzahl an Einträgen kann einer genauso beliebigen Menge zugeordnet werden).

Das Modell kann in verschiedenen Arten erzeugt werden. Zwei häufig auftretende Varianten sind zum einen die Verwendung von für diesen Zweck erstellten Dateien, die das Modell beschreiben, und zum anderen die direkte Verwendung von Klassen im Sourcecode mithilfe von Anmerkungen. Beide dieser Arten haben vor und Nachteile, besonders auch der der Verwendung innerhalb eines Frameworks.

Wenn das Modell mit eigenen Dateien beschrieben wird ist es einfacher zunächst die theoretischen anforderungen an das Modell umzusetzen. Bei nichttrivialen Projekten ist oft das Modell bereits durch die verschiedenen Diagramme

gegeben. Zukünftige Änderungen lassen sich leichter durchführen, wenn das Modell extra definiert wird.

Wenn die Beschreibung des Modell direkt in den Klassen entsteht spart man an Entwicklungszeit.

Moment der Persistierung

Ein weiteres Unterscheidungsmerkmal ist der Moment der Persistierung, d. h. der Zeitpunkt, zu dem neue Änderungen an den Daten in der Datenbank oder ähnlichem abgepeichert werden. Auch hier gibt es in der Regel zwei unterschiedliche Vorgehungsweisen.

Eine Art ist es, die Änderungen erst mit einem bestimmten Methodenaufruf abzuspeichern. Hierdurch wird die Anzahl an Datenbankaufrufen minimiert, da alle Änderungen ïn einem Rutsch"verarbeitet werden.

Die andere Art ist es, verwendete Objekte stets synchron mit der Datenbank zu halten und jede Änderung direkt zu persistieren. Dadurch steigt zwar die Belastung aud der Persistenzebene, dafür muss man als Entwickler weniger auf den aktuellen Zustand achten.

Identifizierung von Dateneinträgen

Bei der Arbeit mit zentralen Server und mehreren Klientensystemen ist die Identifizierung von Dateneinträgen äußerst wichtig.

6 Core Data

Core Data ist ein direkt von Apple entwickeltes Framework zur Verwaltung der Model Ebene. Es findet Verwendung in OS X und iOS Anwendungen und ist aus dem Enterprise Objects Framework entstanden.

6.1 Aufbau

6.1.1 Modellerzeugung

Abstrac Enti- ties, In- herit- tan- ce, Fet- ched Pro- per- ties

Die Erzeugung eines Schemas für die Model Ebene geschieht mithilfe von Datamodel Dateinen mit der *.xcdatamodelid Endung. Xcode bietet wenn ein neues Projekt angelegt wird die Möglichkeit alle benötigten Dateien und Einstellungen automatisch zu erzeugen. Außerdem können die Model-Dateien mithilfe des eingebauten Data Model editor bearbeitet werden.

Ein Model ist aufgebaut aus verschiedenen Entities, deren Eigenschaften und Relationen. Der Aufbau ähnelt dem standardmäßigen Tabellenaufbau.

Eine Eigenschaft einer entity besteht aus einem Namen und einem Typ. Es ist möglich, dem Wert der Eigenschaft Ober- und Untergrenzen zu setzen. Zusätzlich besteht die Möglichkeit zu bestimmen, ob ein Wert gesetzt sein muss oder nicht. Eigenschaften, die nicht in der Datenbank gespeichert werden sollen können als flüchtig markiert werden.

Auch Relationen haben einen Namen und Typ und können optional oder flüchtig sein. Neben des Typ muss auch das Ziel der Beziehung festgesetzt werden. Die Verbindung kann auf ein einzelnes (to-one) oder mehrere (to-many) Objekte zeigen.

6.1.2 Managed Objects

7 REALM Mobile Database