Microservice Architektur: Ein Fundament für mobile Applikationen?

Der renommierte Software-Architekt Martin Fowler hat 2014 den Begriff „Microservice Architektur“ geprägt [4]. Mit einem solchen Architektur-Ansatz wird eine komplexe Anwendungssoftware nicht nur in Software-Komponenten aufgeteilt, sondern diese bleiben auch im Betrieb unabhängige Komponenten, die in einem eigenen Prozess laufen und die untereinander lediglich über technologie- und sprachunabhängige Schnittstellen kommunizieren. Die lose Koppelung der Komponenten bietet diverse Vorteile. Deshalb haben in der Zwischenzeit verschiedene grosse Firmen wie Amazon oder Netflix diesen Architekturansatz aufgegriffen. Auch wir haben uns bei der Überarbeitung des hochschuleigenen Informationssystems „App4Technik“ entschlossen, die Gesamtapplikation aus einer monolithischen Software-Architektur in eine Microservice Architektur zu überführen, die sowohl die Server-Applikation wie auch die mobile Technik-App umfasst. In diesem Artikel beschreiben wir unsere Erfahrungen aus dieser Migration.

Jürg Luthiger | juerg.luthiger@fhnw.ch

„App4Technik“ ist ein verteiltes Software-System bestehend aus:

- verschiedenen Modulen bzw. Server-Services wie Menüplan der Mensa, Neuigkeiten aus der Hochschule für Technik, zukünftige Events der Hochschule oder Beschreibung der Studiengänge;
- einem Content Management System (CMS) für die Marketing-Abteilung zur Pflege der News, der Events und den Informationen zu den Studiengängen;
- einer Android- und einer iPhone-App für die Studierenden und die Mitarbeitenden der Hochschule.

Der Server basiert auf einer monolithischen Software-Architektur, d.h. alle Services laufen im gleichen Prozess. Die Services sind in Java implementiert und sie greifen auf die gleiche Datenbank zu. Sie stellen für das CMS und die mobilen Clients gemeinsam je eine entsprechende Programmierschnittstelle (API) zur Verfügung, die über das HTTP-Protokoll genutzt werden kann (Abb.1). Das API orientiert sich am REST-Programmierparadigma [3].

Die bisherigen mobilen Applikationen sind hybride Apps, d.h. dass sie auf einer gemeinsamen JavaScript-Codebasis basieren und mit der PhoneGap-Technologie als Android- oder iOS-App ausgeliefert werden können. Das ganze Informationssystem ist seit 2013 in Betrieb. Die mobilen Applikationen stehen über die beiden App Stores „Google Play“ und „App Store“ zur Installation bereit.

Der produktive Betrieb des Systems hat Verbesserungspotential aufgezeigt, vor allem in Bezug auf eine einfache Wart- und Erweiterbarkeit. Ebenfalls führt jede Anpassung bei den mobilen Applikationen zu einem erneuten Einreichen und der damit verbundenen Überprüfung der Apps in den entsprechenden App Stores. Da dieser Genehmigungsprozess vor allem bei Apple aufwändig ist, werden Erweiterungen eher zögerlich vorgenommen.

Wir haben uns deshalb entschlossen, die bestehende Applikation im Rahmen einer Bachelor-Arbeit durch die beiden Informatik-Studenten Damian Keller und Matthias Giger zu überarbeiten. Das Ziel der Arbeit ist wie folgt festgelegt worden: „Die bestehende Applikation mit dem monolithischen Ansatz soll in eine Microservice Architektur überführt werden, um den Betrieb und die Erweiterung der einzelnen Dienste zu erleichtern. Dabei ist ein Konzept für die gesamte Applikation zu entwickeln, das eine einfache Integration neuer Dienste und einen einfachen Release-Wechsel bestehender Dienste erlaubt.“ Am Einsatz von PhoneGap soll festgehalten werden. Daher werden folgende Technologien für die neue mobile Applikation vorgegeben:

- *PhoneGap* [6]: PhoneGap ist ein Framework, um mit Web-Technologien wie HTML, CSS und JavaScript mobile Applikationen erstellen zu können. Der grosse Vorteil von PhoneGap ist die Möglichkeit auf Basis eines gemeinsamen Quellcodes mobile Applikationen für unterschiedliche Plattformen generieren zu lassen.
- *Ionic* [5]: Ein Framework für hybride mobile Applikationen mit einem starken Fokus auf das User Interface und die User Interaktion. Ionic Applikationen sollen wie native Applikationen erscheinen. Das Framework arbeitet nahtlos mit PhoneGap und AngularJS zusammen.
- *AngularJS* [1]: Ein modernes, ausgereiftes JavaScript Framework von Google für Rich Internet Applications. Das „App4Technik“-CMS ist bereits mit AngularJS programmiert worden.

Mit der Migration auf die Microservice Architektur versprechen wir uns eine stark verbesserte Flexibilität im Umgang mit bestehenden und neuen Services. Vor allem dann, wenn es uns gelingt diese Flexibilität bis in die mobilen Applikationen hinaus zu führen.

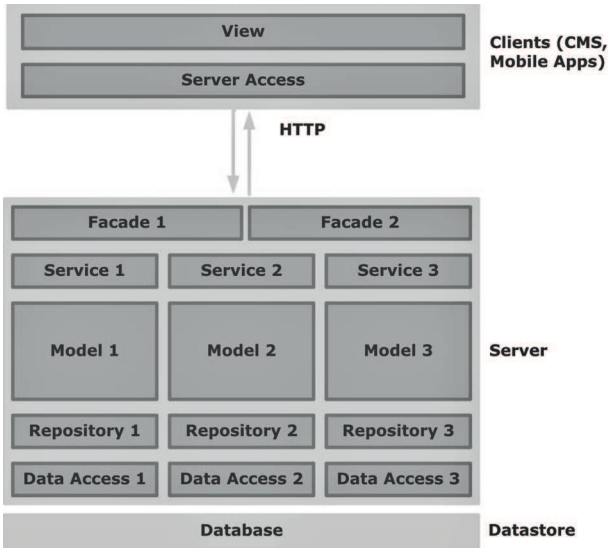

Abbildung 1: Gesamtsystem mit Server als Monolith, separater Datenbank und den Clients (CMS, mobile Applikationen)

Was sind Microservices?

Auf Wikipedia wird der Begriff Microservices wie folgt beschrieben: „Microservices sind ein Architekturmuster der Informationstechnik, bei dem komplexe Anwendungssoftware aus kleinen, unabhängigen Prozessen komponiert werden, die untereinander mit sprachunabhängigen Programmierschnittstellen kommunizieren. Die Dienste sind klein, weitgehend entkoppelt und erledigen eine kleine Aufgabe. So ermöglichen sie einen modularen Aufbau von Anwendungssoftware."

Dank der starken Entkoppelung zeichnen sich Microservices durch folgende Eigenschaften aus:

- Ein Microservice setzt genau einen konkreten Anwendungsfall um.
- Ein Microservice kann einen eigenen Technologie-Stack oder Datenbank nutzen.
- Ein Microservice kann in einer beliebigen Programmiersprache implementiert werden.
- Ein Microservice kann unabhängig von anderen Microservices produktiv betrieben werden.
- Ein Microservice kann einfach ersetzt werden, ohne dass die Gesamtapplikation ausser Betrieb genommen werden muss.

Im Vergleich dazu führt eine enge Koppelung bei einer monolithischen Software-Architektur zu Server-Services mit folgenden Bedingungen:

- Ein Ersatz oder Update eines Server-Services führt zu einer kurzfristigen Nichtverfügbarkeit der Gesamtapplikation, da die Applikation heruntergefahren werden muss, um die neue Funktionalität in Betrieb nehmen zu können;
- Alle Server-Services müssen mit der gleichen Programmiersprache implementiert werden, da sie im gleichen Prozess laufen;
- Alle Server-Services müssen den gleichen Technologie-Stack nutzen.

Die Unabhängigkeit der Microservices untereinander führt zu einer hohen Flexibilität, welche die Microservice-Architektur sowohl für Software-Entwickler wie auch für Software-Betreiber interessant macht.

Konzept für eine flexible „App4Technik"

Um die Microservices bei „App4Technik" einführen zu können, muss zuerst ein neues Architektur-Konzept entwickelt werden, das auf den bestehenden Server-Services und auf den Ideen der Microservice-Architektur basiert.

Wie in Abb. 1 ersichtlich, kann auch mit einer monolithischen Software-Architektur eine saubere Trennung der Anwendungsfälle implementiert werden. Denn jeder Server-Service setzt einen konkreten Anwendungsfall wie Mensa-Service oder News-Service um. Die Server-Services basieren dabei auf einer gängigen Schichten-Architektur mit Service-Layer, den Entitäten als Model und Repository- bzw. Data Access-Layer für den Zugriff auf die Datenbank. Es ist offensichtlich, dass diese Server-Services sehr gute Kandidaten für jeweils einen Microservice darstellen.

Erweitert man nun jeden Server-Service mit einem eigenen REST-API sowie einer eigenen Datenbank und betreibt man diese Server-Services in unabhängigen Prozessen, so ist ein erster Schritt der Entkoppelung vollzogen. Über das REST-API können beliebige Client-Applikationen die Server-Services nutzen. In diesem Projekt sind es zwei: das CMS und die mobile Applikation.

Das CMS ist eine Rich Internet Application (RIA), die für die Marketing Abteilung eine moderne, interaktive Benutzeroberfläche bereitstellt. Im Betrieb hat sich diese Applikation als sehr robust und problemlos wart- und bedienbar erwiesen. Sie wird deshalb nicht auf eine Microservice-Architektur umgeschrieben, sondern als eigenständige Applikation belassen, die über HTTP und über die entsprechenden APIs den Inhalt der Microservices pflegt.

Die mobile Applikation hingegen soll in die Struktur der Microservice-Architektur überführt werden. Es ist wichtig, dass die visuelle Repräsentation und die Funktionalität eines Microservices auf der mobilen Seite über den Microservice selber festgelegt werden kann, so dass der Server und der mobile Teil eine logische Einheit bilden. Zum Beispiel soll ein Microservice nach einer Deaktivierung nicht mehr in der mobilen Applikation erscheinen. Die visuelle Client-Komponente des Microservice hat demnach eine enge Koppelung zum Microservice auf dem Server, auch wenn die beiden Komponenten physisch auf anderen Devices und in unterschiedlichen Prozessen laufen. Logisch gehören sie zusammen.

Zusätzlich müssen die Microservices eine Möglichkeit erhalten, auch untereinander Informationen austauschen zu können. In der Tabelle 1 sind die möglichen Kommunikationsarten aufgelistet.

Aus der Tabelle 1 ist ersichtlich, dass eine one-to-many Kommunikation nur mit einem asynchronen Ansatz realisierbar ist. Eine asynchrone Kommunikation erfordert den Einsatz einer aufwändigen Messaging Infrastruktur, während eine synchrone one-to-one Kommunikation auf Basis des Request/Response Modells von HTTP einfach realisierbar ist.

Um die Kommunikationsart festlegen zu können, ist die bestehenden Applikation analysiert worden. Die Analyse hat gezeigt, dass insgesamt 9 Microservices notwendig sind und dass diese untereinander teilweise abhängig sind. Die Tabelle 2 fasst die notwendigen Microservices zusammen und zeigt auch deren Beziehungen zueinander auf.

Die Tabelle 2 zeigt, dass die Koppelung zwischen den Microservices gering ist. Auch der Informationsaustausch zwischen den Microservices ist einfach und hat eindeutige Kommunikationspartner. Deshalb genügt das einfache, synchrone Request/Response-Kommunikationsmodell von HTTP.

Aus diesen Erkenntnissen kann ein Konzept für den Microservice abgeleitet werden. In Abbildung 2 ist dieses Konzept und die interne Struktur unseres Microservices mit den 3 grösseren Software-Komponenten *Service Consumer*, *Service Server* und *Service Datastore* dargestellt. Der *Service Consumer* ist die mobile Komponente und kommuniziert mit seinem *Service Server* über HTTP. Dieser wird mit einer Schichten-Architektur weiter strukturiert. Eine eigene Datenbank ist für die persistente Datenhaltung der Model-Entitäten zuständig. Für die Kommunikation mit anderen Microservices wird ein HTTP-Client als Gateway in die Server-Komponente integriert.

	one-to-one	one-to-man
synchron	Request/Response	—
asynchron	Point-to-Point	Publish/Subscribe

Tabelle 1: Typische Kommunikationsarten

Die Gesamtfunktionalität von „App4Technik“ wird über verschiedene Microservices realisiert, die auf der Server-Seite einzeln betrieben werden können. Auf der mobilen Seite jedoch, müssen die verschiedenen *Service Consumer* zu einer Einheit zusammenfasst werden, so dass der User die Applikation „App4Technik“ über eine einzige App nutzen kann. Es ist deshalb eine weitere Software-Komponente notwendig, die über ein Framework eine beliebige Anzahl von *Service Consumer* aufnehmen kann (siehe Abb. 3).

Microservices auf dem Server

Die Microservices für „App4Technik“ werden auf dem Server als *Spring Boot* Applikationen implementiert. *Spring Boot* ist ein Framework für die einfache Entwicklung eigenständig lauffähiger Spring-Anwendungen die per „Convention over Configuration», d.h. ohne explizite Konfiguration auskommen und die alle notwendigen Klassenbibliotheken mitbringen, so dass die eigentliche Implementierung erheblich reduziert werden kann.

Die implementierten *Service Servers* nutzen die Unterstützung von *Spring Boot* in grossem Masse. Deshalb wird in diesem Kapitel die Umsetzung der Anwendungsfälle nicht vertieft aufgezeigt. In-

	Microservice	Beschreibung	Nutzt
1	Mensa	Dieser Service stellt der App die Menus der aktuellen Woche zur Verfügung und überprüft, ob von der SV-Schnittstelle neue Menus bereitgestellt werden.	Extern
2	News	Liefert Neuigkeiten inklusive einem Bild pro Neuigkeit. Ausserdem ist es möglich, via CMS die Nachrichten zu verwalten und über eine Push-Nachricht die Nutzer auf eine Nachricht hinzuweisen.	6, 7, 8
3	Events	Speichert Ereignisse, welche wie Neuigkeiten gepusht werden können. Wie bei den anderen Services wird bei Änderungen am Inhalt via CMS das Token beim Security Service geprüft.	6, 7, 8
4	Study Courses	Dieser Service liefert eine Liste von Studiengängen inklusive einem Bild pro Studiengang. Auch hier besteht die Möglichkeit der Push-Notifikation.	6, 7, 8
5	Contacts	Als einfachster Service wird hier nur ein in HTML-formatierter Text mit Kontaktdaten geliefert, welcher bequem über das CMS bearbeitet werden kann.	6, 7, 8
6	Security	Authentifizierung der CMS-Anfragen.	—
7	Devices	Registriert Geräte für Push-Notifikationen und verschickt Notifications.	Extern
8	Images	Speichert und veröffentlicht hochgeladene Bilder.	—
9	Statistics	Liefert Statistiken.	1, 2, 3, 4

Tabelle 2: Liste der Microservices und ihre Abhängigkeiten

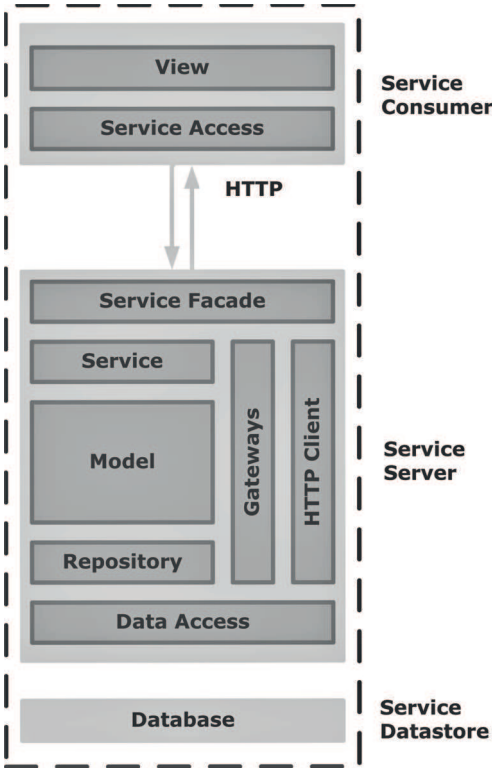

Abbildung 2: Konzept eines Microservices basierend auf einer Schichten-Architektur und HTTP Client als Gateway für die Kommunikation mit anderen Microservices.

	HTTP	URL	Beschreibung
1	GET	/	Returns a list of event objects.
2	GET	/page?page=nr	Returns a list of event objects corresponding to the given page number.
3	GET	/archive?page=nr	Returns a list of expired event object corresponding to the given page number.
4	GET	/{id}	Returns the event object with the given id.
5	POST	/	Creates a new event object.
6	PUT	/{id}	Updates existing event object with given id.
7	DELETE	/{id}	Deletes existing event object with given id.
8	GET	/push/{id}	Pushes event object with given id to all registered devices.
9	GET	/statistics	Returns a statistics representing the download of the event objects.

Tabelle 3: REST-API des Events-Microservice

teressierte finden in der Referenzdokumentation von *Spring Boot* [8] genügend Informationen um die verschiedenen Aspekte der Implementation nachvollziehen zu können.

Wichtiger für das Verständnis der Gesamtapplikation sind die APIs, um zu verstehen wie ein Service Server von einem Service Consumer genutzt werden kann. Wie im Konzept dargelegt, stellen die Server-Services ein REST-API über die Service Facade bereit. Der Events-Microservice zum Beispiel implementiert ein API gemäss Tabelle 3.

Die Informationen, die zwischen *Service Consumer* und *Service Server* fliessen, werden als JSON Objekte codiert. Das Listing 1 zeigt beispielhaft die Implementation der REST-Methode #1 aus

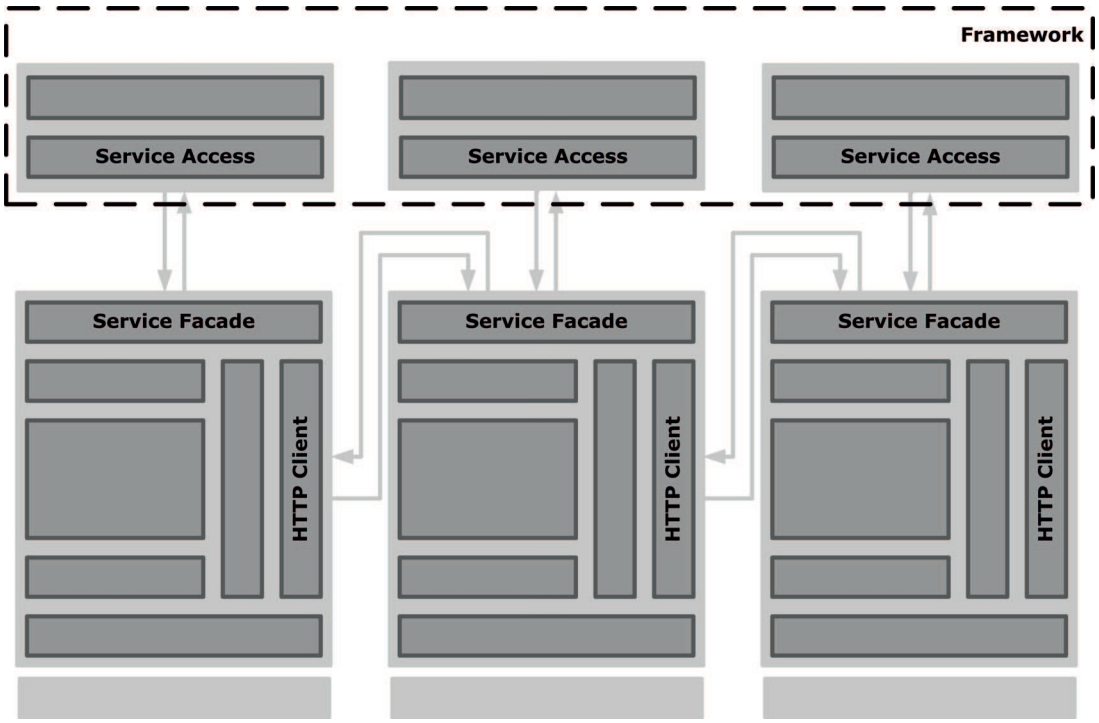

Abbildung 3: Konzept der gesamten Applikation, der Interaktionen und des Frameworks auf der mobilen Seite

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public ResponseEntity<Map<String, Object>> getAllEvents() {                    #1
   Map<String, Object> resultObject = new HashMap<String, Object>();
   List<Event> events = eventsService.loadCurrentEvents();                     #2

   resultObject.put("events", events);                                         #3
   resultObject.put(ConstPool.TIME, Calendar.getInstance().getTime());
   resultObject.put(ConstPool.COUNT, events.size());
   return new ResponseEntity<Map<String, Object>>(resultObject, HttpStatus.OK);  #4
}
```

Listing 1: Auszug aus der Klasse EventsController, welche die Service Facade für den Events Service darstellt.

```
{
   "server_time":1447323320433,
   "count":16,
   "events":[
      {
         "id":53,
         "headline":"Infoveranstaltung",
         "subline":"Master of Science in Engineering…"
  …
```

Listing 2: Auszug aus der JSON Response

der Tabelle 3. In #1 wird als Antwort ein *ResponseEntity*-Objekt erwartet. Das Objekt besteht aus einer Map und dem HTTP Statuscode wie aus #4 ersichtlich ist. In #2 werden die Events aus dem Service *eventsService* geladen und anschliessend in die Map eingefügt (#3). Zusätzliche Informationen ergänzen die Antwort. *Spring Boot* wandelt diese Antwort in einen JSON-String um (siehe Listing 2), der von einem *Service Consumer* gelesen und verarbeitet werden kann.

Kommunikation unter Microservices

Microservices können auch die Dienste anderer Microservices nutzen. Deshalb muss ein Microservice die Möglichkeit erhalten, HTTP-Requests auf das entsprechende REST-API auslösen und eine JSON-Antwort gemäss Listing 2 behandeln zu können.

Aus der Tabelle 2 sieht man, dass zum Beispiel der Events-Microservice den Devices-Microservice nutzt, um über diesen Push-Notifikationen an die registrierten mobilen Geräte senden zu können. Diese Interaktion mit dem Devices-Microservice wird über einen REST-Request ausgelöst, wie in Listing 3 dargestellt.

In #1 wird die Event-Entität, die mittels Push-Notifikation angekündigt werden soll, über den Service aus der Datenbank gelesen. Als HTTP-Client kommt das REST-Template des Spring Frameworks zum Einsatz (#2). Die entsprechende Anfrage wird in #3 aufgebaut und an die URL *pushUrl* gesendet. Die Antwort des Devices-Microservice kann anschliessend überprüft werden, um daraus eine entsprechende Response an den Aufrufenden zu generieren.

```
@RequestMapping(value = "/push/{id}", method = RequestMethod.GET)
public ResponseEntity<Map<String, Object>> push(@PathVariable("id") Integer id) {
   Map<String, Object> res = new HashMap<String, Object>();
   Event event = eventsService.getById(id);                                    #1
   RestTemplate restTemplate = new RestTemplate();                             #2
   HashMap<String, Object> obj = new HashMap<String, Object>();

   obj.put("moduleId", "events");
   obj.put("contentId", event.getId());
   obj.put("title", "Events");
   obj.put("message", event.getHeadline());
   ResponseEntity<Void> response = restTemplate.postForEntity(pushUrl, obj, Void.class);    #3

   if (response.getStatusCode() == HttpStatus.OK) {
      res.put(ConstPool.PUSH_DATE, now);
      return new ResponseEntity<Map<String, Object>>(res, HttpStatus.OK);
   } else {
      return new ResponseEntity<Map<String, Object>>(HttpStatus.INTERNAL_SERVER_ERROR);
   }
}
```

Listing 3: Kommunikation zwischen Microservices

Abbildung 4: Launcher Screen auf der mobilen App mit fünf Modulen. Ein Modul entspricht einem Microservice. Das Modul wird über ein Icon repräsentiert, das vom Microservice bereitgestellt wird.

Microservices auf dem mobilen Client

Microservices werden auf der Server-Seite dank Frameworks wie Spring Boot sehr effizient unterstützt. In unserem Projekt wollen wir das Konzept der Microservices aber bis in den mobilen Client weiterziehen. Der Microservice soll auch auf der mobilen Seite seine visuelle Repräsentation (siehe Abb. 4) und seine Funktionalität festlegen können.

Zudem muss auch der Zeitpunkt, wann ein Microservice in Betrieb genommen wird, flexibel gehalten werden können. Das hat zur Folge, dass die mobile Applikation eine Möglichkeit umsetzen muss, um Module laden und darstellen zu können, die zu einem späteren Zeitpunkt in Betrieb genommen werden. Um diese Anforderung realisieren zu können, müssen die folgenden zwei Bedingungen erfüllt sein:

- *Service Discovery:* Das REST-API eines Microservices wird über eine URL identifiziert und kann beliebig gestaltet werden, da der Microservice auf irgendeinem Server betrieben werden kann. Um den Microservice nutzen zu können, muss ein Client herausfinden können, wo sich der Microservice befindet und wie er angesprochen werden kann.
- *Dynamic Service Loading:* Die Funktionalität eines beliebigen Microservices ist bei der Implementation der mobilen Applikation nicht bekannt. Deshalb muss die mobile Applikation ein Framework bereitstellen, das ein zusätzliches Nachladen der entsprechenden Funktionalität zu einem späteren Zeitpunkt erlaubt. Das dynamische Hinzufügen neuer Funktionen in eine mobile Applikation wird aber von den Betreibern der App Stores sehr unterschiedlich unterstützt. Während Google diesbezüglich sehr offen ist, weist die Firma Apple in ihren Richtlinien sehr hohe Einschränkungen aus. So schreiben sie in [2] folgendes: “Apps that download code in any way or form will be rejected” und “Apps that install or launch other executable code will be rejected”.

Unsere Technik-App ist eine hybride App, die auf Basis der Web-Technologien HTML, CSS, JavaScript programmiert ist. Neue Funktionalität wird demnach als JavaScript Code nachgeladen, was auch bei jeder mobilen, modernen Website der Fall ist. Deshalb ist die Chance gross, dass Apple das *Dynamic Service Loading* von JavaScript Code nicht abweist – was sich auch als wahr erwiesen hat.

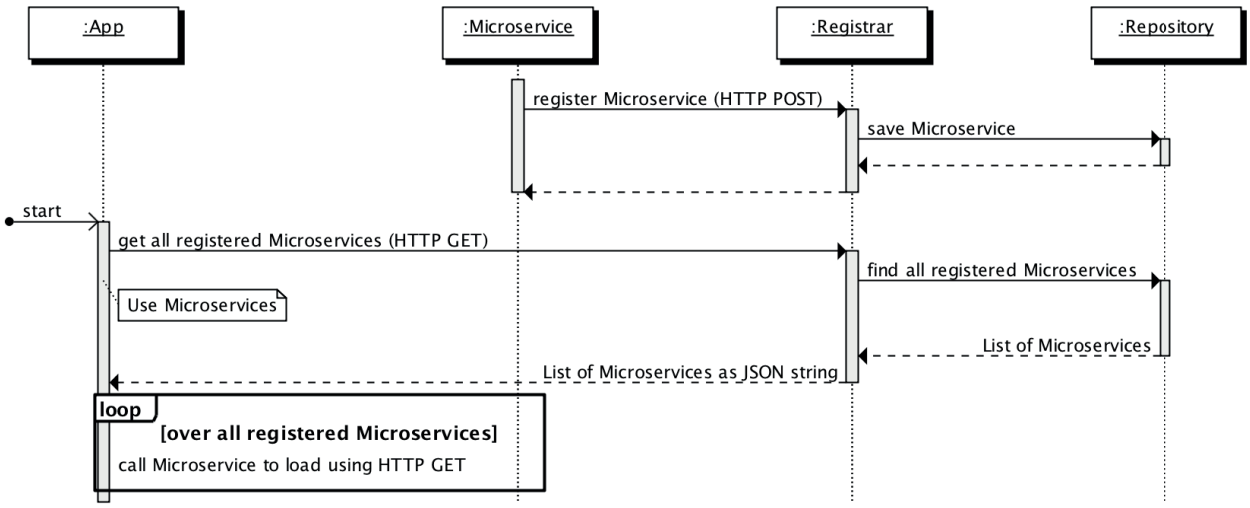

Abbildung 5: Das Sequenzdiagramm zeigt das Konzept für Service Discovery und die Interaktion verschiedener Aktoren (App, Microservice) mit der Registrierungsstelle Registrar

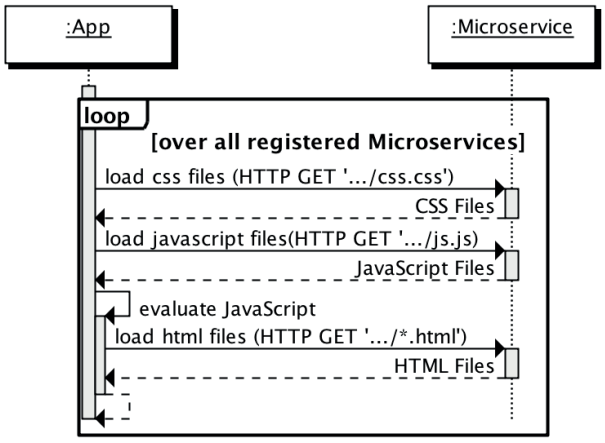

Abbildung 6: Sequenzdiagramm des dynamischen Ladens des mobilen Teils des Microservices

	HTTP	URL	Beschreibung
1	GET	/js.js	Der Code des Microservice für die mobile App, beinhaltet die Funktionalität auf der Client-Seite
2	GET	/css.css	Die CSS-Stylesheets für das Design des mobilen Microservices
3	GET	/icon.svg	Icon für den Launcher Screen
4	GET	/icon.png	Alternative Variante des Icons für den Launcher Screen, falls das SVG-Format nicht unterstützt wird
5	GET	/check	Statusabfrage

Tabelle 4: REST-API für das dynamische Nachladen

Das *Service Discovery* kann mit dem Design Pattern *Client-side Service Discovery* umgesetzt werden [7]. Dabei gibt es einen bekannten Ort, wo die Microservices registriert werden und wo ein Client die Zugangsinformationen zu den einzelnen Microservices abfragen kann. In Abbildung 5 ist der Ablauf eines Registrierungsschrittes und einer entsprechenden Abfrage in einem vereinfachten Sequenzdiagramm aufgezeichnet. Die Registrierungsstelle wird als Microservice implementiert und unter einer öffentlichen URL betrieben. Der Administrator dieses zentralen Microservices ist befugt, weitere Services zu registrieren und zu verwalten.

Für das *Dynamic Service Loading* müssen die Microservices eine einheitliche Schnittstelle umsetzen, so dass ein Client die fehlende Benutzerschnittstelle und die fehlende Funktionalität als HTML, CSS und JavaScript Files sowie Icons vom entsprechenden Microservice nachladen kann. Der Microservice selber muss diese Dateien als statische Ressourcen bereitstellen.

In Abb. 6 ist das Konzept dieses dynamischen Ladens abgebildet. Die aufgeführten REST-Abfragen legen das REST-API fest, welches jeder Microservice unterstützen muss, um die Integration in die mobile Applikation ermöglichen zu können. Über eine HTTP-GET Anfrage werden die fehlenden CSS- und anschliessend die JavaScript-Files geladen. Der JavaScript-Code wird sofort ausgeführt, was wiederum einen HTTP-Request auf den Server für zusätzliche statische Ressourcen wie HTML-Files oder Icons auslösen kann.

Um das *Dynamic Service Loading* unterstützen zu können, muss ein Microservice zusätzlich zu seinem API aus Tabelle 3 weitere Methoden öffentlich anbieten. Diese sind in Tabelle 4 zusammengefasst.

Der zentrale Code für das Nachladen ist in Listing 5 abgebildet. In Zeile #1 erfolgt der HT-

```
// Load and add CSS Styles
$http.get(url + 'css.css').success(function(data) {                    #1
   loadCSS(data);
});

// Load and execute JavaScript
$http.get(url + 'js.js').success(function(data) {                      #2
   eval(data);                                                         #3
});
```

Listing 5: Zentrale Funktionalität für das dynamische Laden von CSS und JavaScript-Code

```
/@RequestMapping(value = "/css.css", method = RequestMethod.GET)       #1
public Resource css() {
   return new ClassPathResource("style.css");                          #3
}

@RequestMapping(value = "/js.js", method = RequestMethod.GET)          #2
public Resource js() {
   return new ClassPathResource("controller.js");                      #4
}
```

Listing 6: Server Methode für das Laden des CSS und JavaScript Files

TP-GET Request auf die CSS-Files. Anschliessend kann der JavaScript-Code geladen (#2) werden. Dieser wird in der Zeile #3 ausgeführt. Daraus können weitere HTTP-Anfragen resultieren.

Listing 6 zeigt die entsprechenden Server Methoden. In Zeile #1 und #2 wird das Mapping auf die HTTP-Anfragenfestgelegt und entspricht den Anfragen aus Listing 5. Das Laden der entsprechenden Dateien wird in Zeile #3 und #4 ausgeführt.

Fazit

Mit dem Umbau der App haben wir uns eine stark verbesserte Flexibilität im Umgang mit den einzelnen Modulen versprochen. Vor allem wollten wir die Möglichkeit erhalten, während dem Betrieb der Applikation „App4Technik" Module zu ersetzen, neue Module hinzuzufügen oder Module entfernen zu können. Mit der gewählten Microservice-Architektur ist es uns gelungen, diese Anforderungen erfolgreich umzusetzen.

Der Aufwand des Umbaus hat sich für uns aber nur deswegen gerechtfertigt, weil die Betreiber der App-Stores, insbesondere Apple-Store, das dynamische Nachladen vom JavaScript-Code unterstützen. Denn Microservice-Eigenschaften wie Unterstützung unterschiedlicher Technologien oder unterschiedlicher Programmiersprachen haben in unserem Kontext eher geringe Priorität. Das Studententeam hat aber zu Demonstrationszwecken Microservices in PHP implementiert und erfolgreich in das System integriert.

Da die Abhängigkeit der Microservices untereinander gering ist, kann die Kommunikationsinfrastruktur auf der Basis des einfachen HTTP-Request/Response-Modells aufgebaut werden. Deshalb ist die Komplexität eines einzelnen Microservice nicht hoch, einfach zu überblicken und gut verständlich. Fehlerbehebungen oder Ergänzungen können effizient realisiert werden.

Ein Microservice stellt in diesem Projekt ein kleines verteiltes Client-Server-System dar. Die Server-Komponente kann mit Hilfe geeigneter Frameworks wie *Spring Boot* sehr schnell implementiert und sehr gut mit automatisierten, funktionalen Tests validiert werden. Die Client Komponente hingegen wird im Kontext einer mobilen Applikation betrieben. Deshalb ist das Testen dieser Komponente anspruchsvoller und führt zu aufwändigeren Integrationstests, die nicht einfach zu automatisieren sind.

Für den Betrieb der gesamten Applikation „App4Technik" stehen verschiedene Optionen offen. Einerseits kann jeder Microservice über einen eigenen Webserver verfügen und in einem eigenen Prozess auf unterschiedlichen Servern laufen und andererseits können alle Microservices als Webmodule in den gleichen Webserver gepackt und über die Administrationsmöglichkeiten des Webserver separat administriert werden oder man wählt eine Kombination beider Varianten, was in unserem Betrieb aktuell der Fall ist.

Referenzen

[1] AngularJS: https://angularjs.org

[2] Apple. "App Store Review Guidelines": https://developer.apple.com/app-store/review/guidelines, 2015.

[3] Fielding, Roy. "Architectural Styles and the Design of Network-based Software Architectures", Dissertation, 2000.

[4] Fowler, Martin. "Microservices": http://martinfowler.com/articles/microservices.html, 2015

[5] Ionic: http://ionicframework.com

[6] PhoneGap: http://phonegap.com

[7] Richardson, C. "Client-side service discovery": http://microservices.io/patterns/client-side-discovery.html, 2014.

[8] Spring Boot: http://projects.spring.io/spring-boot