Hauptprojekt

Alexander Piehl
alexander.piehl@haw-hamburg.de

Hamburg University of Applied Sciences,
Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany

1 Einleitung

2 REST

REST ist ein Architekturstil für verteilte Systeme. Die Abkürzung REST steht für Representational State Transfer [CK09]. Erstmals wurde REST 2000 von Roy Fielding vorgestellt [KLC13]. Die Rest-Architektur wird häufig für Client-Server Anwendungen verwendet. Dabei ist REST zurzeit sehr belie bbei der Entwicklung von Webservices, da REST Webservices wohl nicht nur leichter zu implementieren sind, sondern auch einfacher zu skalieren sind. [CK09]. Unter Anderem aus diesen Gründen stellten Google, Facebook und Yahoo ihre Services von SOAP auf REST um [Rod08, NCH+14].

REST basiert dabei auf Resource Oriented Architecture, kurz ROA [CK09]. Dies bedeutet, dass jede wichtige Information als Ressource zur Verfügung stehen muss [PR11]. Die Zugänglichkeit zu der Ressource muss über eine eindeutige URI gegeben sein. Die Ressourcen sollen zusätzlich über verschiedene Methoden manipuliert werden können. Es müssen mindestens die sogenannten CRUD-Operatoren zur Verfügung stehen. CRUD steht für Create, Read, Update und Delete und beschreibt die grundsätzlichen Daten Operationen. Bei Rest werden dafür die standardisierten HTTP-Methoden verwendet, welche im Standard RFC 2616 definiert wurden sind [KLC13]. In der Tabelle 1 werden die Beziehung zwischen den CRUD Operatoren und den HTTP Operatoren dargestellt.

CRUD-Operation	HTTP-Methode
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Tabelle 1. Beziehung CRUD und HTTP Operatoren [RVG10]

Neben den CRUD-Operatoren können noch weitere HTTP-Methoden zur Verfügung stehen, wie z.B. HEAD und OPTIONS [PR11].

Die jeweiligen Aufrufe müssen Zustandslos erfolgen [RVG10, PR11, KLC13]. Im Detail heißt dies, dass der Webservices keine Informationen über den Zustand seiner einzelnen Clients speichert. Sollten Informationen über den Zustand notwendig sein, müssen die Clients die Informationen mitgeben. Dahingehend ist es auch mit REST möglich kompliziertere Programmzustände abzubilden [PR11].

Die jeweiligen Nachrichten können in verschiedene Formate vorliegen [RVG10]. Sehr häufig werden XML oder JSON oder beide Formate für die Nachrichten verwendet. Eine Vorschrift existiert nicht.

Zusammenfassend lässt sich REST in vier Grundprinzipien zusammenfassen [PR11]:

- **Addressability:** Jede wichtige Informationen muss als Ressource vorliegen und über eine eindeutige URI erreichbar sein.
- **Connectedness:** Die Repräsentation der Ressourcen ist getrennt von den Ressourcen. Dies bedeutet Ressourcen können in verschiedenen Formate vorliegen, wie JSON und XML.
- **Uniform Interface:** Auf die Ressourcen wird nur über standardisierten HTTP-Methoden zugegriffen.
- **Statelessnes:** Jede Kommunikation erfolgt Zustandslos.

Webservices, welche die vier Grundprinzipien einhalten, bekommen häufig den Beinamen RESTful [PR11]. Der Begriff RESTful ist jedoch nicht eindeutig definiert.

2.1 Besonderheiten beim Testen ?

3 Consumer Driven Contract Test

Bei dem Ansatz Consumer-Driven Contracts werden die Schnittstellen bzw. die Verträge aus der Sicht des Clients/Consumers definiert. Dabei spielt es keine Rolle, ob es nur einen Cosumer oder mehrere existieren. Die jeweiligen Anfragen und gewünschten Antworten werden notiert und auf Grundlage dieser Informationen wird der Service implementiert.

Wie sich aus der generellen Beschreibung dieses Ansatzes ableiten lässt, funktioniert der Ansatz nur, wenn sowohl die Consumer wie der Service neu implementiert werden. Bei bestehenden Anwendungen könnte dieser Verfahren dafür genutzt werden, um zu überprüfen, ob der Service unötige Schnittstellen anbietet.

Auf der Basis dieses Konzeptes setzen die Consumer Driven Contract Tests auf. Ein Tool, welches Consumer Driven Contract Tests unterstützt ist PACT. PACT ist ein Open-Source Tool, welches auf Github verwaltet wird.

In der Ausführung von Toby Clemson über das Testen von Microservices wird dieser Verfahren explizit empfohlen. Dabei benennt er neben PACT noch zwei weitere Tools, welche Consumer Driven Contract Tests anbieten. Diese Tools sind PACTO und Janus. Für das Hauptprojekt wurde sich für PACT entschieden, da nach der ersten Recherche PACT den größten Umfang bietete und zum PACT besser dokumentiert ist, als die anderen Tools.

- Erklärung Consumer Driven Ansatz
  - Nutzer einer Schnittstelle definiert den Contract
  - Da der Nutzer wohl besser weiß, was er nutzen möchte und was nicht
  - Service Provider können nur erahnen
  - Vorteile es ist klar definiert, was der Service nutzen soll. Daher kann er so schlank wie möglich implementiert werden
  - Vorteile dazu gibt es einen feinjörnigen Einblick und schnelles Feedback für das planen von Änderungen. Ergänzend dazu können gezielt einzelnen Consumer angesprochen werden.
  - Aufgrundlage der Consumer Verträge wird der Service erstellt
- Erläuterung Consumer Driven Contract Test
  - Tool PACT
  - Consumer definiert seine Anfrage und die zu erwarteten Antworten, samit UNIT-Test
  - Implmentierung im Consumer
  - Beim Ausführen dieser Tests wird ein Server gestartet, der mit den entsprechenden Antworten auf die Anfragen reagiert
  - Dabei wird ein File erstellt, welches die Aufrufe und Antworten enthält
  - mit diesem File wird nun der Service Provider getestet
  - Antwortet er auf die Anfragen korrekt, wie es im File beschrieben ist

3.1 Warum Consumer Driven Contract Test

- Vorteile bei Änderungen im Service kann sofort wieder gegen die Datei getestet werden, um zu prüfen, ob alle Consumer noch korrekt funktionieren
- übersicht, was die Consumer Anfragen
- sachnelles Feedback
- wissen, ob Änderung zu ungewollten Fehlern
- wissen, welche consumer aktualisiert werden müssen
- MARS System, welches regelmäßig angepoasst wird
- Architektur, welche Fokus auf des Netzwerk hat
- Fehler wegen Schnittstelle problematisch

Was verspricht man sich davon? Verbindung zu MARS - System im Wandel

3.2 Implementierung

3.3 Einbindung in MARS

- Aussuchen eines Services und Consumers
- Aufrufen herausfinden via Browser und Postman
- Testen der Aufrufe via Postman, um festzustellen das sie funktionieren und koirrekt sind
- Aktueller Stand als korrekt anzusehen und Soll-Zustand
- Implementierung des Consumers

3.4 Fazit

- Erklärung
- Erläuterung zu der Verbindung mit Microservice
- Normale Implementierung
- Einbinden in Mars
- Fazit

3.5 Fazit

4 Ausblick Masterarbeit

Literaturverzeichnis

[CK09] Sujit Kumar Chakrabarti and Prashant Kumar. Test-the-rest: An approach to testing restful web-services. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World:*, pages 302–308. IEEE, 2009.

[FB15] Tobias Fertig and Peter Braun. Model-driven testing of restful apis. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1497–1502. ACM, 2015.

[KLC13] Chia Hung Kao, Chun Cheng Lin, and Juei-Nan Chen. Performance testing framework for rest-based web applications. In *2013 13th International Conference on Quality Software*, pages 349–354. IEEE, 2013.

[NCH+14] Alvaro Navas, Pedro Capelastegui, Francisco Huertas, Pablo Alonso-Rodriguez, and Juan C Dueñas. Rest service testing based on inferred xml schemas. *Network Protocols and Algorithms*, 6(2):6–21, 2014.

[PR11] Ivan Porres and Irum Rauf. Modeling behavioral restful web service interfaces in uml. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1598–1605. ACM, 2011.

[Rob06] Ian Robinson. Consumer-driven contracts: A service evolution pattern, 2006.

[Rod08] Alex Rodriguez. Restful web services: The basics. *IBM developerWorks*, 2008.

[RVG10] Hassan Reza and David Van Gilst. A framework for testing restful web services. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 216–221. IEEE, 2010.