HTBLuVA St. Pölten
Höhere Abteilung für Informatik

DIPLOMARBEIT

Einsatz von Java FX

im Projekt Eventtechnik

Ausgeführt im Schuljahr 2018/19 von:

Matej Bradaric, 5AHIF-02
Lukas Heinzl, 5AHIF-05
Oliver Hovorka, 5AHIF-06
Maximilian Suchy, 5AHIF-20

Betreuer/Betreuerin:

OSTR Mag. Otto Reichel

St.Pölten, am 19. November 2018

Inhaltsverzeichnis

Kapitel 1

Einleitung

Zur Entstehung dieses Kapitels wurde folgende Quelle verwendet:
[Architectural Styles and the Design of Network-based Software Architectures]

1.1 Entstehung von REST

Das Ziel dieser Diplomarbeit ist es diverse REST-APIs aus zu werten und zu entscheiden welches für das Projekt CarSharing am besten geeignet ist. Der Architektur Stil 'Representational State Transfer' (REST) wurde im Jahre 2000 von Dr.Roy Fielding in seiner Dissertation 'Ärchitectural Styles and the Design of Network-based Software Architectures" hergeleitet. Er bezieht sich in seiner Arbeit auf diverse Netzwerk-basierte Architektur Stile, aus welchen er dann seinen eigenen Stil ableitet. Dr.Fielding legt in seiner Arbeit diverse Prinzipien fest, denen man in der REST Architektur folgen muss.

1.1.1 Client-Server-Architektur

Das erste Prinzip das Dr.Roy Fielding festlegt ist das sogenannte Client-ServerPP-rinzip. Dies besagt das man die Server Implementation und die des Clients zu 100 Prozent trennt, die Trennung dieser Zwei zusammenarbeitenden Systeme bringt mehrere Vorteile. Der erste Vorteil den man erlangt ist die höhere Portabilität des Clients. Der zweite Vorteil ist die Möglichkeit die beiden Komponenten absolut voneinander getrennt weiter entwickeln zu können.

1.1.2 Zustandslosigkeit

Das zweite Prinzip welches Dr.Roy Fielding definiert ist das Zustandslosigkeitsprinzip, welches besagt, dass jede Anfrage die an den Server gesendet wird alle Informationen zum Verarbeiten dieser Anfrage beinhalten muss. Der Server darf nicht auf gespeicherte Daten zugreifen um eine Antwort zu senden. Hiermit werden folgende Eigenschaften verbessert: Sichtbarkeit, Zuverlässigkeit und Skalierbarkeit.

1. **Sichtbarkeit:**
   Die Sichtbarkeit wird durch die Informationskapselung gesteigert, dies gelingt dadurch, dass jede Anfrage nun separat betrachtet werden kann. Durch diese Separation der Anfragen ist es möglich zu garantieren, dass sich die Anfragen nicht gegenseitig beeinflussen.

2. **Zuverlässigkeit:**
   Die Zuverlässigkeit wird durch die Sicherheit vor Teil-Ausfällen gesteigert. Dies wird gewährleistet, da andere Teile des Systems die Aufgaben des ausgefallenen Teils übernehmen können.

3. **Skalierbarkeit:**
   Die Skalierbarkeit wird verbessert da der Server die Zustandsdaten der einzelnen Anfragen nicht speichern oder verwalten muss. Dadurch kann er schneller wieder Ressourcen freigeben und die Implementation wird wesentlich erleichtert, da man kein Ressourcenmanagement zwischen den Anfragen bewältigen muss.

Jedoch hat die Zustandslosigkeit nicht nur Vorteile, sondern auch einen großen Nachteil nämlich die Erhöhung des Netzwerkverkehrs. Dieser erhöht sich dadurch dass nichts gespeichert wird und viele Daten bei jeder Anfrage erneut gesendet werden müssen.

1.1.3 Cache

Um dem eben zuvor genannten Problem der Datenvermehrung entgegen zu wirken, können Daten als 'Cacheable' oder 'Non-cacheable' definiert werden. Bei 'Cacheable' Daten ist dem Client oder dem Server erlaubt die Daten zu speichern. Dies ermöglicht es dem Benutzer eine kürzere Latenzzeit zu gewährleisten, währenddessen auch der Netzwerkverkehr verringert wird. Das Speichern der Daten bringt jedoch ein Problem mit sich, es kann nämlich passieren dass die Daten bereits veraltet sind. Daher kann die Konsistenz des Systems nicht völlig gewährleistet werden. Es liegt daher am Entwickler die richtigen Daten als 'Cacheable' zu klassifizieren.

1.1.4 Einheitliche Schnittstellen

Das Ziel des REST-Architektur Stiles ist es die einzelnen Komponenten voneinander zu trennen und die Datenlieferung so einheitlich wie möglich zu gestalten. Dies vereinfacht die Implementierung wesentlich und weiters können Teile des Systems separat für andere Systeme verwendet werden. Hier ist wiederum auch ein Problem versteckt, nämlich wird die Effizienz des Systems verringert. Da eben alle Daten immer einheitlich gesendet werden, kommt es zu dem Mitsenden von ungenützten Daten.

1.1.5 Code-On-Demand

Diese Funktionalität von REST ist für die Vereinfachung der Entwicklung von Clients zuständig. Es wird den Entwicklern ermöglicht bereits fertige Codesegmente zu verwenden. Dadurch werden Fehlerquellen deutlich reduziert und die Implementierung erfolgt wesentlich schneller. Hierbei handelt es sich um die einzige optionale Funktionalität von REST. Dr. Roy Fielding begründet dies so:

> 'The notion of an optional constraint may seem like an oxymoron. However, it does have a purpose in the architectural design of a system that encompasses multiple organizational boundaries. It means that the architecture only gains the benefit (and suffers the disadvantages) of the optional constraints when they are known to be in effect for some realm of the overall system'.
> [Architectural Styles and the Design of Network-based Software Architectures]

1.2 Gliederung

Diese Arbeit ist folgender Maßen gegliedert:
Kapitel 1 - Einleitung: In der Einleitung wird beschrieben was genau ein REST-API überhaupt ist.
Kapitel 2 - Kriterien
Kapitel 3 - Bewertungen
Kapitel 4 - Ergebnisse
Kapitel 5 - Praktische Umsetzung im Projekt
Kapitel 6 - Konklusion und Zukunftsaussichten

Kapitel 2

Kriterien

In diesem Kapitel wird festgelegt welche genauen Kriterien bei jedem API untersucht und bewertet werden. Es wird zu jedem API zuerst eine Auswertung der Kriterien und danach eine Benotung geben. Weiters wird bestimmt wie die einzelnen Noten (1-4) strukturiert sind. Die Kriterien werden aufgeteilt in 3 Kategorien: Allgemeines, Entwicklung. In der Kategorie Allgemeines wird behandelt ob es Hilfestellungen (Tutorials, Codeexemplare, usw.) gibt und es wird der momentane Zustand des APIs beschrieben. Die Kategorie Entwicklung beschäftigt sich damit wie die Entwicklungsumgebungen der diversen APIs aufgebaut sind. Ebenfalls werden hier die Software und Hardware Voraussetzungen für die diversen APIs beschrieben.

2.1 Allgemeines

1. Gibt es Hilfen zum erlernen des APIs?
   Wenn ja, welche sind vorhanden?
2. Wie gut ist das API Dokumentiert?
3. Braucht man eine Lizenz für das API?
   Wenn ja, unter welchen Kosten ist das API verfügbar?
4. Gibt es eine Community für dieses API?
   Wenn ja, ist diese Community aktiv und/oder hilfsbereit?

2.2 Entwicklung

1. Welche Entwicklungsumgebungen gibt es für das API?
2. Kann man mit dem API einen Standalone - Server programmieren?
3. In welchen Betriebssystemen beziehungsweise in welchen Umgebungen kann der Server ausgeführt werden?
4. Was für Hardware benötigen die Server um zu Funktionieren?
5. Welche Java Version ist die minimale die von dem API unterstützt wird?
6. Gibt es zur Hilfe bei der Modellierung Grafische Tools?
7. Ist es möglich mit dem API einen Client zu entwickeln?
   Wenn ja, gibt es Möglichkeiten sich Teile des Clients generieren zu lassen?
8. Welche Android SDK ist die minimale die von dem API unterstützt wird?
9. Bietet das API spezielle Funktionalitäten für das Projekt CarSharing?

2.3 Benotung

Das Ziel dieser Diplomarbeit ist es das beste API für das Projekt CarSharing aus zu werten. Daher wird jede Benotungskategorie mit einer Gewichtung unter besonderer Beachtung des Projekts versehen. Abhängige Kindkategorien werden jedoch nicht gewichtet und bekommen auch keine Note, da sie beim Elternpunkt mit in die Noten einbezogen werden. Es gibt auch zu jeder Gewichtung eine Beschreibung warum diese so gewählt wurde. Diese Gewichtung wird sich im Rahmen von 1-5, wobei 5 das beste ist, bewegen. Bei der Bewertung der einzelnen APIs wird jede Kategorie eine Note bekommen. Diese Noten werden sich ebenfalls von 1-5, wobei 5 hier auch das beste ist, bewegen. Weiters wird es zu jeder Kategorie eine Analyse geben aus welcher die Noten dann hervorgeht. Nach der Benotung der einzelnen Kategorien werden alle Noten eines APIs mit der Gewichtung der jeweiligen Kategorie multipliziert und daraus ergibt sich dann eine Gesamtnote für das API.

2.3.1 Gewichtung

Tabelle 2.1: Allgemein

Nr.	Bezeichnung	Gewichtung	Begründung
1	Gibt es Hilfen zum erlernen des APIs?	2	Bei dieser Kategorie ist die Gewichtung eher niedrig angesetzt da bereits Vorwissen zu gewissen APIs aus dem Unterricht besteht.
2	Wie gut ist das API Dokumentiert?	5	Diese Kategorie wurde sehr hoch gewichtet, da die Dokumentation eines APIs extrem wichtig ist um das API zu verstehen.
3	Braucht man eine Lizenz für das API?	4	Hier wurde die Gewichtung auch relativ hoch gewählt, da das Budget für das Projekt CarSharing sehr minimal bis nicht existent ist.
4	Gibt es eine Community für dieses API?	3	Bei dieser Kategorie ist die Gewichtung mittig angesetzt, da einem eine Community bei Problemen jeglicher Art immens weiterhelfen kann.

Tabelle 2.2: Entwicklung

Nr.	Bezeichnung	Gewichtung	Begründung
1	Welche Entwicklungsumgebungen gibt es für das API?	3	Bei dieser Kategorie ist die Gewichtung mittig angesetzt, da es eine wenige wirklich schlechte Entwicklungsumgebungen gibt jedoch genau so wenig wirklich gute.
2	Kann man mit dem API einen Standalone - Server programmieren?	5	Hier wurde die Gewichtung sehr hoch angesetzt, denn es würde das Projekt wesentlich mühsamer machen wenn der Server immer von etwas abhängig wäre.
3	In welchen Betriebssystemen beziehungsweise in welchen Umgebungen kann dieser ausgeführt werden?	4	Diese Kategorie wurde hoch gewichtet, da der bereits vorhandene Server Linux verwendet. Daher wäre es mühsam einen Server mit einem anderen Betriebssystem.
4	Was für Hardware benötigen die Server um zu Funktionieren?	2	Bei dieser Kategorie ist die Gewichtung niedrig angesetzt, da der bereits vorhandene Server ziemlich gute Hardware besitzt. Daher ist es eher unwahrscheinlich, dass ein Upgrade notwendig ist.
5	Welche Java Version ist die minimale die von dem API unterstützt wird?	1	Hier ist die Gewichtung sehr niedrig festgelegt, da das Projekt in einer sehr neuen Java Version programmiert wird. Es wäre daher nur wichtig für die Rückwertskompatibilität und das ist auch eher irrelevant, da die Software nur auf neueren Geräten laufen wird.
6	Gibt es zur Hilfe bei der Modellierung Grafische Tools?	3	Diese Kategorie ist mittig gewichtet, da es einerseits nicht viel Modellierung bedarf in dem Projekt CarSharing und andererseits ist es immer gut sich Software zuerst grafisch zu Visualisieren.

Entwicklung:

Nr.	Bezeichnung	Gewichtung	Begründung
7	Ist es möglich mit dem API einen Client zu entwickeln?	5	Hier ist die Gewichtung sehr hoch ausgefallen, da in dem Projekt CarSharing eine mobile Applikation entwickelt wird welche als Client den REST-Service abfragt.
8	Welche Android SDK ist die minimale die von dem API unterstützt wird?	3	Hier ist die Gewichtung sehr niedrig festgelegt, da das Projekt in einer sehr neuen Android Version programmiert wird. Es wäre daher nur wichtig für die Rückwertskompatibilität und das ist auch eher irrelevant, da die Software nur auf neueren Geräten laufen wird.
9	Bietet das API spezielle Funktionalitäten für das Projekt CarSharing?	5	Bei dieser Kategorie ist die Gewichtung sehr hoch, da eine solche Funktionalität die Entwicklung des Projekts massiv erleichtern würde. Daher wäre ein solche Feature ein wirklich aussagekräftiger Grund dieses API zu wählen.

Note	Punkte
5	202 bis 225
4	180 bis 201
3	157 bis 179
2	135 bis 156
1	112 bis 134

2.3.2 Notenskala

Alle einzeln vergebenen Noten fügen sich nach der Bewertung in eine Gesamtnote zusammen. Hier wird nun aufgestellt ab wie vielen Punkten welche Gesamtnote vergeben wird. Diese Note wird darüber entscheiden welches API in dem Projekt CarSharing verwendet wird. Insgesamt sind 225 Punkte verfügbar, daraus ergibt sich folgende Tabelle:

Literaturverzeichnis

[Architectural Styles and the Design of Network-based Software Architectures]
https://www.ics.uci.edu/ fielding/pubs/dissertation/fielding_dissertation.pdf
REST - Definierende Dissertation
19.11.2018

[Architectural Styles and the Design of Network-based Software Architectures]
https://www.ics.uci.edu/ fielding/pubs/dissertation/fielding_dissertation.pdf
Zitat-Dr. Fielding optionale Funktionalität S.84
19.11.2018