Web-basierte Anwendungen 2:

Verteilte Systeme

Logbuch

von Ushpal Mann

Matrikelnummer: 11076980

Inhaltsverzeichnis

1. Ideenfindung

Innerhalb des Workshops sollten wir ein Projekt Partner finden. Ich und Nazan haben uns entschieden an dem Projekt zusammen zu arbeiten. Da ich mit Nazan, meine Team Kollegin, schon mehrere Projekte in verschiedenen Bereichen absolviert haben und somit ein gutes Team sind, konnten wir direkt mit den ersten Aufgabe “Ideenfindung” beginnen.

Zunächst haben wir uns überlegt was man für das Projekt umsetzen könnte.
Mögliche Projektideen waren :
1. Tic Tac Toe - Spiel
2. Online Katalog für Bibliothek
3. FahrradVerleih

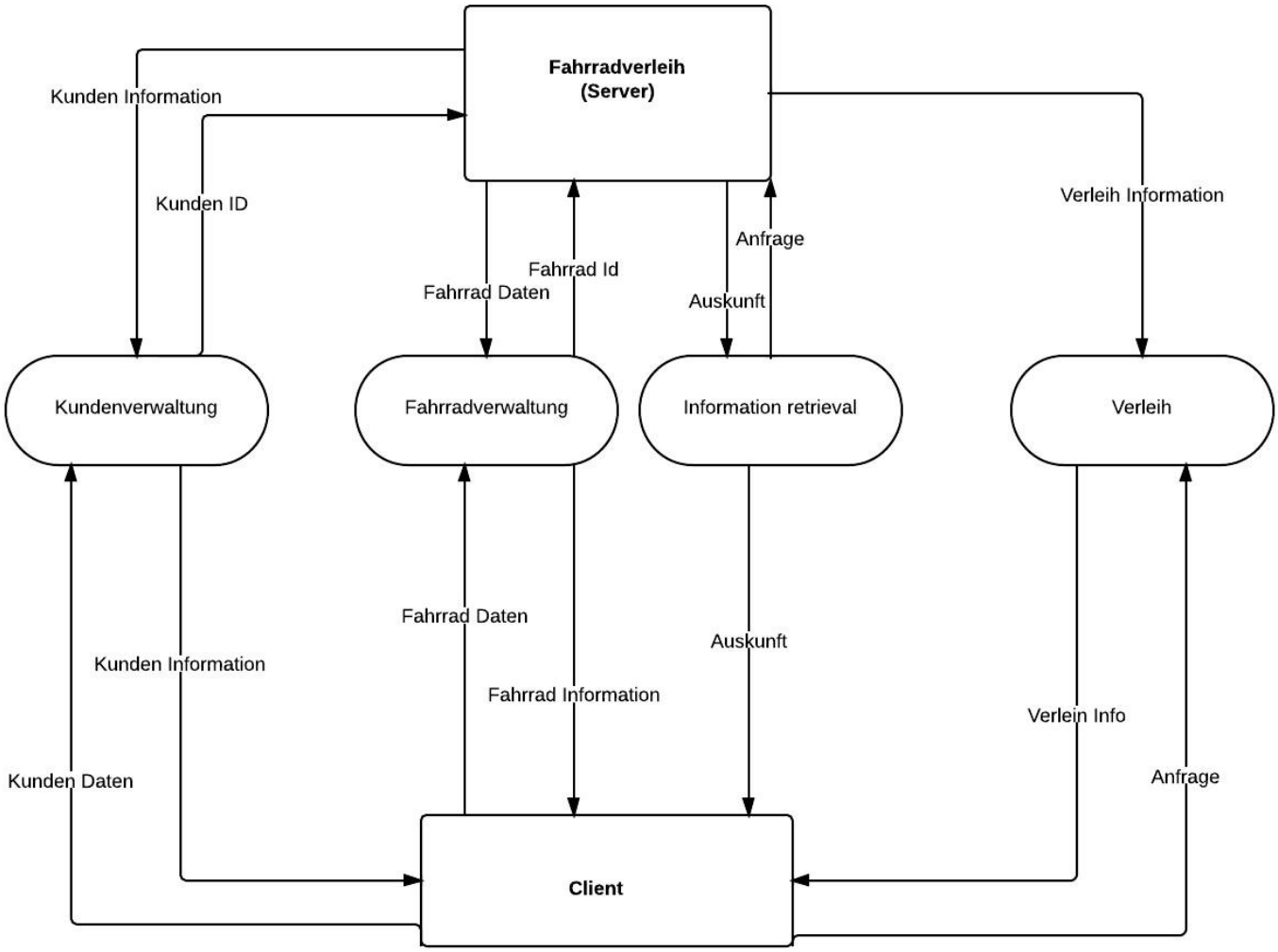
Nach der absprache mit den Mentoren haben wir uns entschieden “Fahrradverleih” als Projekt umzusetzen.
In den Workshop haben wir im Internet nach ähnliche Produkte in Markt recherchiert, um eine Überblick über die wesentliche Funktionen zu schaffen.
Wir haben grobe Funktionen und Eigenschaften für unser Projekt festgelegt.

2. Kommunikationsabläufe und Interaktionen

Wir sollten für unser Projekt ein Informationsflussdiagramm erstellen. Als erstes haben wir im Internet über Informationsflussdiagramme recherchiert. Als Ergebnis haben wir viele verschiedene Arten von Diagrammen bekommen, wie ein Informationsfluss dargestellt werden kann. Wir wussten nicht wie ausführlich der Informationsfluss in dem Diagramm abgebildet werden soll. Ich und Nazan haben zusammen zwei Entwürfe gemacht.

Informationsflussdiagramm - Entwurf

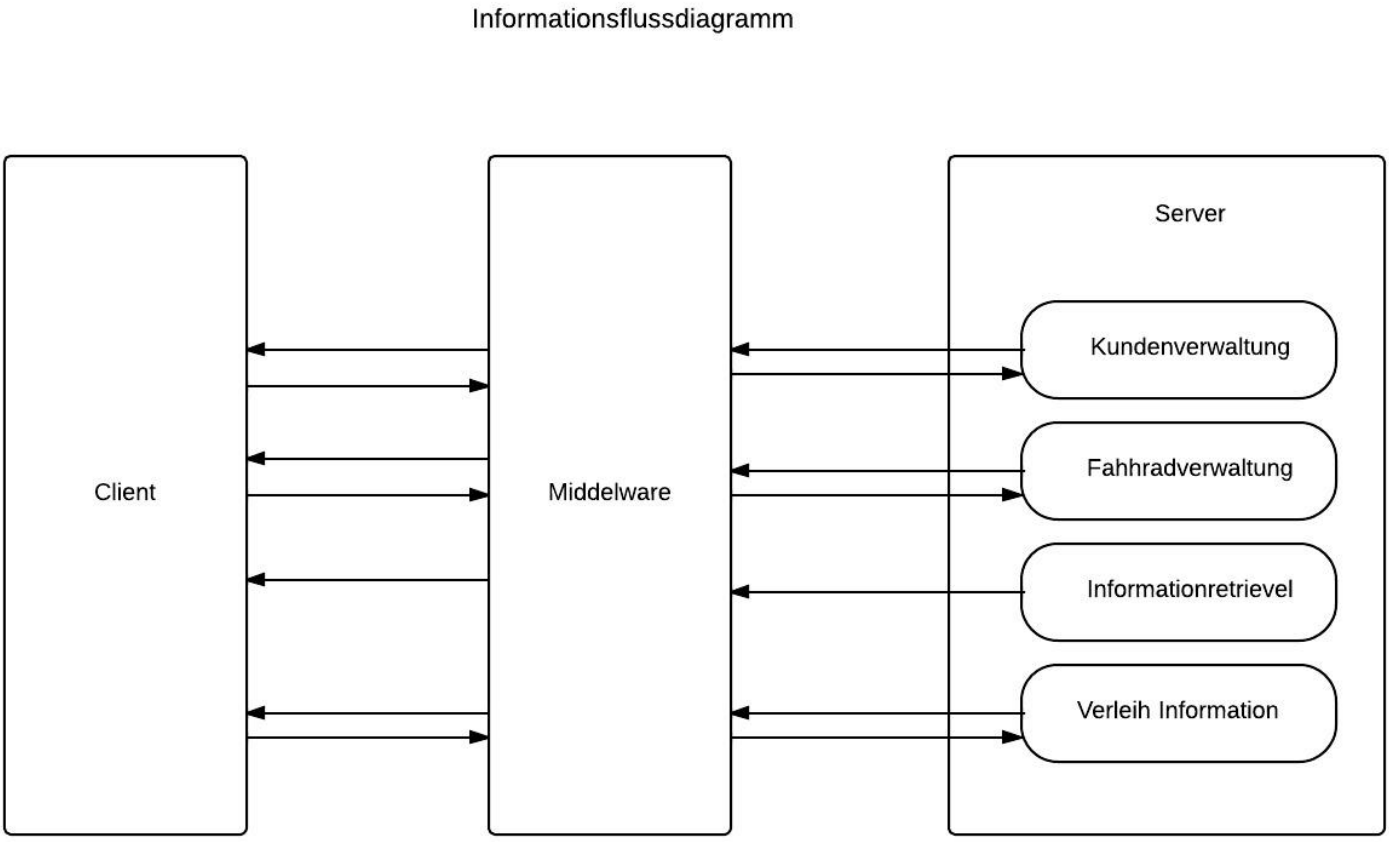
Informationsflussdiagramm



In den Betraungsgeshpräch mit den Mentoren haben wir unsere Entwürfe vorgelegt. Sie haben uns empfohlen ein Diagramm, eine Kollaboration der beiden Entwürfe zu machen und zwischen Client und Server eine Middleware abzubilden, zusätzlich eine Anwendungsfall zu schreiben und diese mit einem Diagramm darzustellen.

Wir haben uns entschieden, dass ich das Informationsflussdiagramm mache und Nazan eine Anwendungsfall mit einem Diagramm macht.

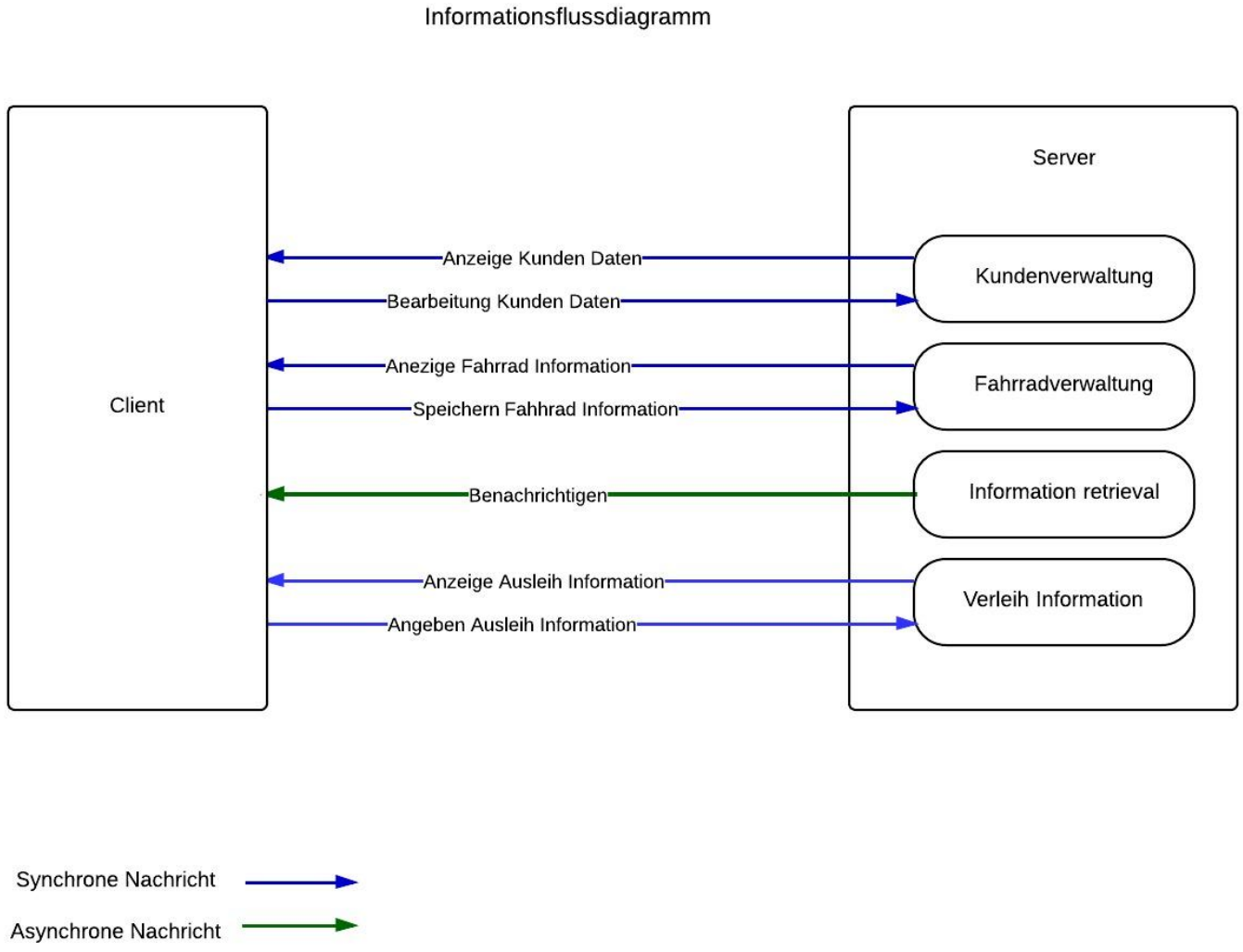
Ich habe folgenden Diagramm erstellt



In den zweiten Beratungsgeschpräch haben wir unseren Informationsflussdiagram diskutiert. Kritik und Verbesserungsvorschäge:

- das Middleware Komponente soll entfernt werden, da es bereits in Client und Server integriert ist.
- Kommunikationsvorgänge beschriften.
- asynchrone und synchrone Nachrichten erkennbar zeichnen.

Ich habe die Kritikpunkte in betracht gezogen und das Diagramm verbessert bzw. geändert.
Endergebnis: Informationsflussdiagramm



3. Projektbezogenes XML Schema

Es sollte ein valides XMLSchema erstellt werden. Wir haben uns gedanken gemacht welche Daten in unsern Projekt benötigt werden und haben einen Schema erstellt.

Link: Erste XML-Schema

Bei dieser Aufgabe hatten wir zuerst einen Lösungsweg gewählt bei dem wir die Informationen (Name, Vorname, etc.) als Attribute mitgegeben haben. Allerdings fanden wir diese Variante unschön, deshalb haben wir uns nach Rücksprache mit den Betreuren dafür entschieden es durch Elemente zu realisieren.
In den Beratungsgeschpräch ist uns auch aufgefallen, dass es viele wichtige Informations Daten gefehlt haben.

In den zweiten Versuch entstand folgende Schema.
Link: XML Schema

Link: XML Datei

In diesen Schema hatten wir einen kritischen Punkt und zwar in der FahrradType wurde eine Verweis auf ein UserType gemacht. Dadurch sind die ganze Informationen von User nochmal in FarradType. Wir wollten das vermeiden, aber wüssten nicht wie man am besten nur auf UserId referenziert.
Wir haben nach Referenzierung in XSD recherchiert und in Büchern nachgeschlagen und haben zwei Varianten gefunden ID/IDREF und KEY/KEYREF.

Durch die ID und Key Ausdrücke wird sichergestellt, dass die User-Id und Fahrrad-Id Nummer eindeutig sind und nicht doppelt vergeben werden können.

ID/IDREF war einfach zu verwenden, jedoch hatte eine/zwei entscheidende Nachteile:
Als UserVerweis können irrtümlich FahrradID Werte (und umgekehrt) eingesetzt werden.
“Die IDs müssen mit Buchstaben beginnen, was die Verwendung als (z.B. per Sequenz generierte) DatenbankID erschwert”[1].

Gegensatz zu ID/IDREF’s wird in KEY/KEYREF verfahren sichergestellt, dass bei Userelementen (UserID="43" FahrradID="151") als UserVerweis nur UserId und als

[1] http://www.torsten-horn.de/techdocs/java-xsd.htm#Schema-keyref

FahrradVerweis nur FahrradidWerte eingesetzt werden können.

KEY/KEYREF hatte wesentliche vorteile gegenüber ID/IDREF. Deswegen habe ich entschieden KEY/KEYREF’s zu verwenden.

Code Ausschnitt

```
<key name="userKey">
        <selector xpath="user/user" />
        <field xpath="@id" />
</key>

<keyref name="userKeyref" refer="userKey">
        <selector xpath="ausleih" />
        <selector xpath="fahrrad" />
        <selector xpath="rückgabe" />
        <field xpath="@userID" />
</keyref>

Fahrrad Id ref

<key name="fahrradKey">
        <selector xpath="fahrrad" />
        <field xpath="@id" />
</key>

<keyref name="fahrradKeyref" refer="fahrradKey">
        <selector xpath="ausleih" />
        <selector xpath="user" />
        <selector xpath="rückgabe" />
        <field xpath="@fahrradID" />

</keyref>
```

In einer Diskussion mit Kommilitonen haben wir erfahren, dass JAXB KEY/KEYREF nicht unterschtützt. Sie haben uns empfohlen ID/IDREF zu verwenden und vor allem zuerst eine XML Datei zu schreiben und anhand dieser eine XML Schema zu erstellen.

Wir haben ein valides XML Datei erstellt und danach ein XML Schema ausgearbeitet in dem wir ID/IDREF's angewendet haben.

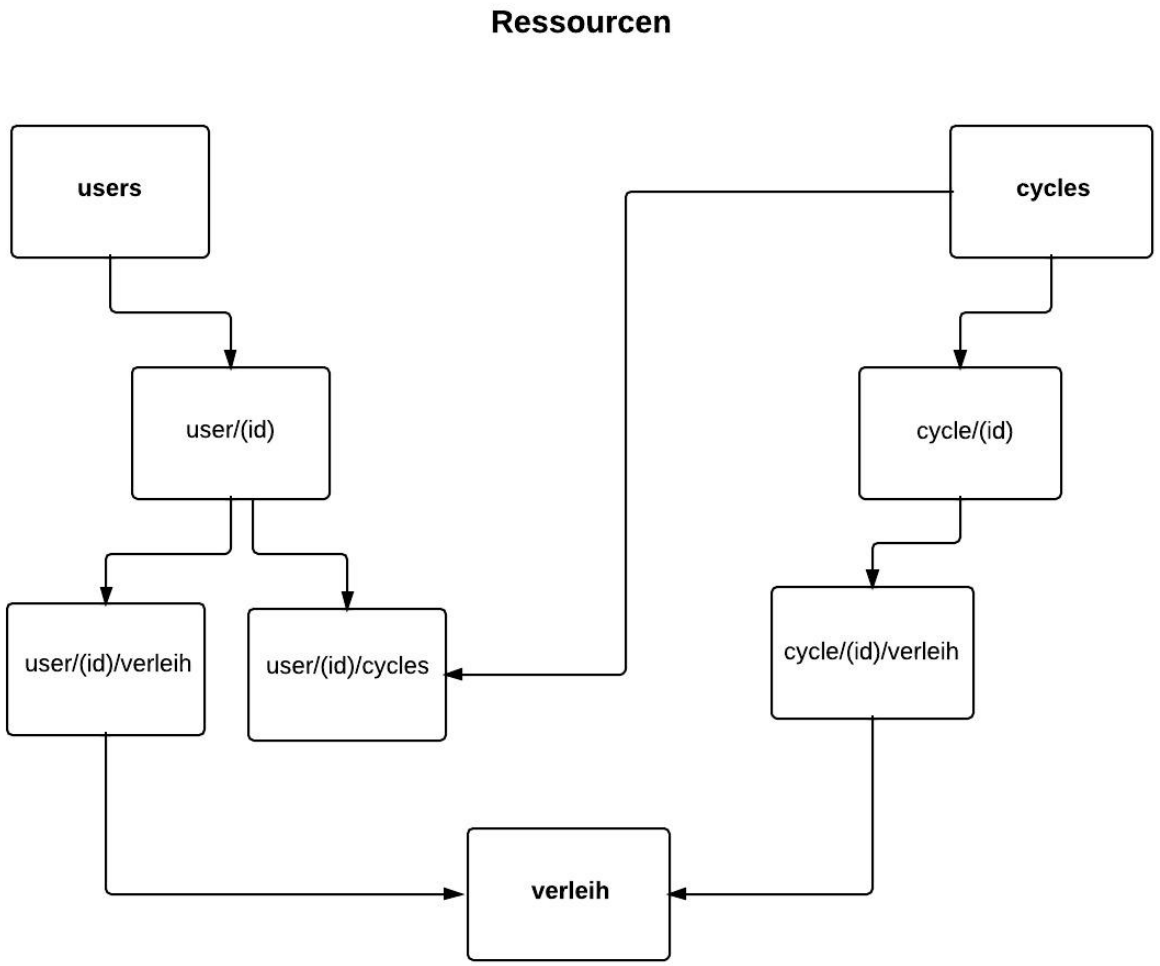
Außerdem, habe ich für Fahrräder Model "Enumeration" verwendet, so dass nur aus bestimmten Werten das Model gewählt werden darf.

Schema mit ID's
Link: XML Schema

4. Ressourcen und die Semantik der HTTP-Operationen für das Projekt

Als nächstes haben wir für unseren Projekt entscheidende Ressourcen bestimmt. Wir haben Users, Fahrräder und Verleih als Hauptressourcen gewählt.
Ich habe eine Ressourcen Diagramm sowie ein Ressourcen Tabelle erstellt. In der Tabelle sind die Ressourcen mit deren URI und Methoden abgebildet

Ressourcen Diagramm



Ressourcen Tabelle

Ressource	URI	Methode
Liste aller Fahrräder	/fahrraeder/	GET, POST
Einzelne Fahrrad	/fahrraeder/fahrrad/(id)	GET, PUT, POST,DELETE
Liste aller User	/users/	GET, POST
Einzelne User	/user/user/(id)	GET,PUT,POST,DELETE
Ausleiher eines Fahrrad	/fahrraeder/fahrrad/(fahrradid)/vergebenAn	GET,PUT,POST,DELETE
Liste aller Fahrräder des Benutzers	/users/user/(id)/fahrraeder	GET,POST
Ausleihe eines bestimmten Fahrrad	/fahrraeder/fahrrad/(fahrradid)/ausleih	GET,PUT,POST,DELETE

5. RESTful Webservice

Wir haben mit Hilfe des JAXB Frameworks aus unseren XML Schema Java Klassen generiert. Zudem haben wir uns an der verlinkten Tutorial Seite für Marshalling und Unmarshalling orientiert. Ich habe eine Main Klasse geschrieben, die unsere XML Datei einlesen und ausgeben sollte.

Beim ausführen der Main Klasse ist immer ein Fehler aufgetreten. Wir haben nach den Fehlermeldung in Internet gesucht, dennoch könnten leider keine Lösung finden.
Wir haben uns an die Mentoren gewendet und sie sind unser XML und XSD durchgegangen und uns darauf hingewiesen, dass wir keine Umlaute (z.b straße, Fahrräder etc) benutzen sollten. Wir haben das XML Schema überarbeitet und die Java Klassen neu erzeugt, aber das Problem bestand immer noch.
Die Tutoren haben uns angeboten, dass wir ihnen unser Code per Mail schicken sollen, was wir sofort gemacht haben.
Ich habe von den Betreuer eine Email erhalten, in dem sie mir ein Lösungsvorschlag genannt haben um den Fehler zu beheben. Es lag in dam Root-Element, da es keine Typ sein darf.

Ich habe das XML-Schema verbessert und ein anderes Design (Venetian Blind Design) gewählt. Vorher hatten wir “Salami Slice Design”.

"Salami Slice Design" ist das wenig geschachtelte Schema mit Referenzen auf viele global deklarierte Elemente und Attribute. Die Kindselemente werden nicht lokal definiert, sondern per “ref” referenziert[2].

Venetian Blind Design hat nur ein einziges global definiertes Element, wodurch es sofort klar wird welches das Rootelement ist. Außerdem die Struktur ist übersichtlich und leicht zu verstehen.

Link: Final XML-Schema

REST

Am Anfang hatte ich keine Kenntnisse darüber wie man Ressourcen implementiert. Für den Einstieg habe ich die Einführungsvideos geguckt und in Büchern nach geguckt. Anhand der

[2] http://www.torsten-horn.de/techdocs/java-xsd.htm#Schema-Salami

Beispiele in den Büchern habe ich begonnen die GET Methoden zu implementieren. Allerdings wusste ich nicht ob ich auf dem richtigen Weg war. Ich habe mich mit meinen alten Kommilitonen in Verbindung gesetzt und um Hilfe gebieten. Sie haben mir einen Denkanstoß gegeben, dass ich für alle generierte Java Klassen Konstruktoren anpassen bzw selber legen soll und für den Zugriff auf die Datenstäze neue Methoden schreiben soll.

Ich habe für jede Klasse selber Konstruktoren und Methoden geschrieben, die in dem Code mit “Eigener Teil” gekennzeichnet ist.

Zwischendrin habe ich bemerkt, dass ID’s den Typ String haben, was nicht optimal für den Vergleich zweier ID’s war. Deshalb habe ich den Typ von “ID” nach Integer geändert.

Mit Hilfe von Komilitionen und zur Verfügung gestellten Links in Wiki, habe ich die ersten GET Methoden in den Service Klasse implementiert. Allerdings den Code für den Grizzly Server habe ich nicht selbst geschrieben, sondern aus dem Links entnommen.

Die GET Methoden habe ich auf den Browser getestet. Danach habe die Methoden (Z.b Get Fahrräder) mit Query Parameter erweitert. Nun könnte man Fahrräder nach Model, Rahmen Nummer, Hersteller und nach FahrradId suchen.
Somit war der Suche auch in den GET Methode darin und müsste nicht seperat implementiert werden. Deswegen habe ich das “Suche” Teil aus dem XSD und Java Klassen rausgenommen.

Während der Implementation der Post-Fahrrad und Post-Verleih ist mir aufgefallen, dass es sich um redundante Daten handelt und das Verleih nicht unbedingt als eigene Ressource dargestellt werden muss. Verleih ist abhängig vom Fahrrad, wenn ein Fahrrad nicht existiert kann auch keine Ausleihe existieren.
Ich habe das XSD und Java Klassen angepasst, in dem ich AusleihType zu FahrradType hinzugefügt habe und VerleihType aus dem Kollektion entfernt habe. Infolgedessen ist unser XSD kompakter und übersichtlicher geworden.

Das weitere Problem bestand darin, dass “MietBeginn” und “MietEnde” in XSD aus dem Typ “dateTime” waren. Nach der konvertierung in Java war der Typ XMLGRegorianCalendar. Ich habe versucht aus dem XMLGRegorianCalendar einen String zu machen, jedoch beim speichern der Ausleihe wurde einen Fehler ausgegeben. Deswegen habe ich fürs Erste MietEnde und MietBeginn den Typ String gegeben.

Die Methoden habe ich mit RestClient getestet. Alle Methoden haben funktioniert bis auf POST Fahrräder. Es wurde immer ein “Internal Server Error” geworfen.

6. Client Entwicklung

Ich hatte keine Erfahrung mit dem Swing Client oder überhaupt mit der Java Client Implementierung.
Mit Tipps und Hilfe von Kommilitonen habe ich einen “Konsolen” basierten Client implementiert, der möglcihe Funktionen des Verleihsystems darstellt.
Mit dem Client ist es möglich:
- User Liste abrufen
- User hinzufügen,
- User Daten ändern,
- User löschen
- ein User mit Id abrufen

sowie
- Fahrrad liste sehen
- einen bestimmten Fahrrad abrufen
- Fahrrad Daten ändern
- Fahrrad löschen

und bei Ausleihe Id der Ausleihenden hinzufügen, ändern und löschen.

Hinzufügen der Fahrräder hat leider nicht funktioniert. Es wurde“Internal Server Error” ausgegeben. Ich habe nach den Fehlermeldung im Internet gesucht und erfahren, dass es ein Fehler beim Speichern von Metadaten sein kann.

Um zu überprüfen, ob es an bestimmten Daten liegt, habe ich versucht wenige Fahrrad Daten zu speichern (Bild und Standort weggelassen -näturlich habe ich die Konstruktoren und die Methoden in dem Web Service auch angepasst). Trotz der Änderung wurde die gleiche Fehlermeldung ausgegeben. Ich könnte die Ursache für der Fehler nicht finden.
In Bezug zu der Fehlermeldung habe ich den Betreuern eine Mail geschrieben (mit dem Anhang:Code).

Außerdem, könnte man die Ausleihe-Information nicht speichern, es wurden immer ein “Not Found” geworfen.

Nun habe ich versucht mit Hilfe von Tutorials im Internet und mit Codebeispielen das Swing Client umzusetzen.Die Schwierigkeit bestand darin es mit Service Methoden zu verbinden.

Einfügen, Ändern und Löschen von Usern funktionierte einwandfrei. Allerdings das PUT, POST

und DELETE von Fahrräder hat nicht funktioniert.

Aufgrund der wenigen Erfahrung in Java und fehlende Zeit könnte ich leider nicht alle Methoden in den “Swing” Client implementieren und XMPP Teil umsetzen.