

Fachhochschule Köln  
Cologne University of Applied Sciences

# Web-basierte Anwendungen 2: Verteilte Systeme

Sommersemester  
2013

## Projektdokumentation

Fahrradverleih

von Nazan Aysune und Ushpal Mann

betreut von  
Prof. Dr. Kristian Fischer

---

## Inhaltsverzeichnis

Seite	Inhalt
3	Einleitung
4	Projektidee und Funktionen
6	Anwendungsfall
7	Informationsflussdiagramm Version 1
8	Erklärung (Informationsflussdiagramm V.1)
9	Informationsflussdiagramm Version 2
10	Erklärung (Informationsflussdiagramm V.2)
10	Informationsflussdiagramm Version 3
11	Informationsflussdiagramm (verbessert)
12	XML-Dokument und Erklärung
13	Überarbeitetes XML-Dokument
15	XML-Schema
18	Erklärung und Kritik/Verbesserung
19	Überarbeitetes XML-Schema
22	Ressourcen und http-Operationen
23	Ressourcen und Methoden
25	Überarbeitete Methoden
26	RESTful Webservice und Methoden
28	Operationen
33	Konzeption und XMPP Einrichtung
33	Client Entwicklung
34	Screenshots (Client)
37	Literaturverzeichnis

---

## Einleitung

In dem Workshop des Faches WBA 2 - Verteilte Systeme wurden in der Phase 2 diverse Aufgaben um den Bereich RESTful Webservices bearbeitet. Die Hauptaufgabe war es den Umgang mit XML-Dateien zu erweitern und die Erstellung eines eigenen Services, welcher auf diese Daten zugreift. Außerdem war die Erstellung eines eignen Servers und Clients gefordert.

In dieser Dokumentation werden die Ergebnisse aufgeführt und erläutert, sowie die Vorgehensweise mit den Zwischenergebnissen und Alternativen aufgeführt.

---

## Projektidee

Die zweite Phase fing damit an, eine angemessenen Projektidee vorzustellen, das als verteiltes System umgesetzt werden kann. Der erste Gedanke war es ein Zweipersonen-Strategiespiel namens "Tic Tac Toe" zu erstellen, das wegen der minderen Komplexität verworfen wurde.

Eine andere Alternative war es einen Online Katalog für eine Bibliothek (Bücher, Video) zu entwickeln, der den Verleih und die Ausleihe organisieren bzw. Benachrichtigungen an die Anwender schicken sollte, jedoch würde diese Art von System zu wenig Gebrauch finden. Desweiteren war die WG-Organisation in Bezug auf die Einkäufe im Gespräch, allerdings ist es nicht ersichtlich gewesen, sinnvolle Funktionen für dieses System festzulegen.

Schlussendlich kam die Idee einen privaten Auto und Fahrradverleih zu konstruieren. Aus Versicherungsgründen wurde der Autoverleih rausgenommen und der Fahrradverleih ausgearbeitet. Folgende Funktionen wurden als Möglichkeit angesehen:

- Eine Online Registrierung um die Nutzung vom System zu ermöglichen
- Anzeige der verfügbaren Fahrräder und des Standortes bzw. Abgabeortes
- Fahrräder können für einen bestimmten Zeitraum reserviert werden
- Der Nutzer kann seinen Account einsehen bzw. bearbeiten
- evtl. kann eine Benachrichtigung geschickt werden, wenn ein Fahrzeug sich in der Nähe befindet.

Die zentrale Aufgabe in dieser Phase ist es festzulegen welche Daten ausgetauscht werden. Diese Daten bilden das Fundament eines Systems. Die Daten werden in die synchrone und asynchrone Kommunikation eingeteilt. Die synchrone Kommunikation bewirkt, dass beim Senden oder beim Empfangen von Daten immer synchronisiert wird, also wird gewartet bis die Kommunikation abgeschlossen ist.

Zu der synchronen Kommunikation in unserem System zählen folgende Daten:

### Online Registrierung (Clientseitig)

- Telefonnummer
- Vorname
- Nachname
- Straße
- PLZ
- Ort
- Land

- 
- E-Mail
  - Zahlungsart (Kreditkarte / Lastschrift)
  - Fahrrad zur Ausleihe vorhanden / nicht vorhanden

Die eingegebenen Daten werden auf Validität überprüft und an den Server geschickt. Anschließend erhält der Sender eine Bestätigung (E-Mail / SMS)

### **Anzeige der verfügbaren Fahrräder und des Ortes**

- Fahrrad im Ort und im bestimmten Umkreis suchen und Anzeigen (Request, Clientseitig)
  - genaue Adresse eingeben (per GPS bestimmen lassen)
  - Umkreis eingeben
  - evtl. Zeitraum eingeben in dem ein Fahrrad benötigt wird (sonst Anzeigen von Fahrrädern, die ganztägig verfügbar sind)
- Fahrrad und Ort Anzeigen (nach erfolgreicher Suche (Response, serverseitig))
  - Fahrrad-ID
  - Besitzer
  - Standort bzw. Abgabeort
  - Verfügbarkeitszeitraum
  - Preis
- Reservierung von Fahrzeugen (Clientseitig)
  - Zeitraumangabe

Die asynchrone Kommunikation besteht im Wesentlichen darin, dass das Senden und Empfangen von Daten zeitlich versetzt ist. Deshalb hat es sich als sinnvoll erwiesen, die folgenden Daten als solches zu definieren.

- 
- Account einsehen bzw. bearbeiten (Clientseitig)
    - persönliche Daten können bei nicht aktiver Ausleihe bearbeitet werden
  - Benachrichtigungen verschicken (Serverseitig)
    - Wenn sich ein verfügbares Fahrrad in der Nähe befindet wird eine Nachricht per E-Mail, SMS oder als Push-Benachrichtigung an den User gesendet
  - Bewertung von Usern (positiv / negativ)

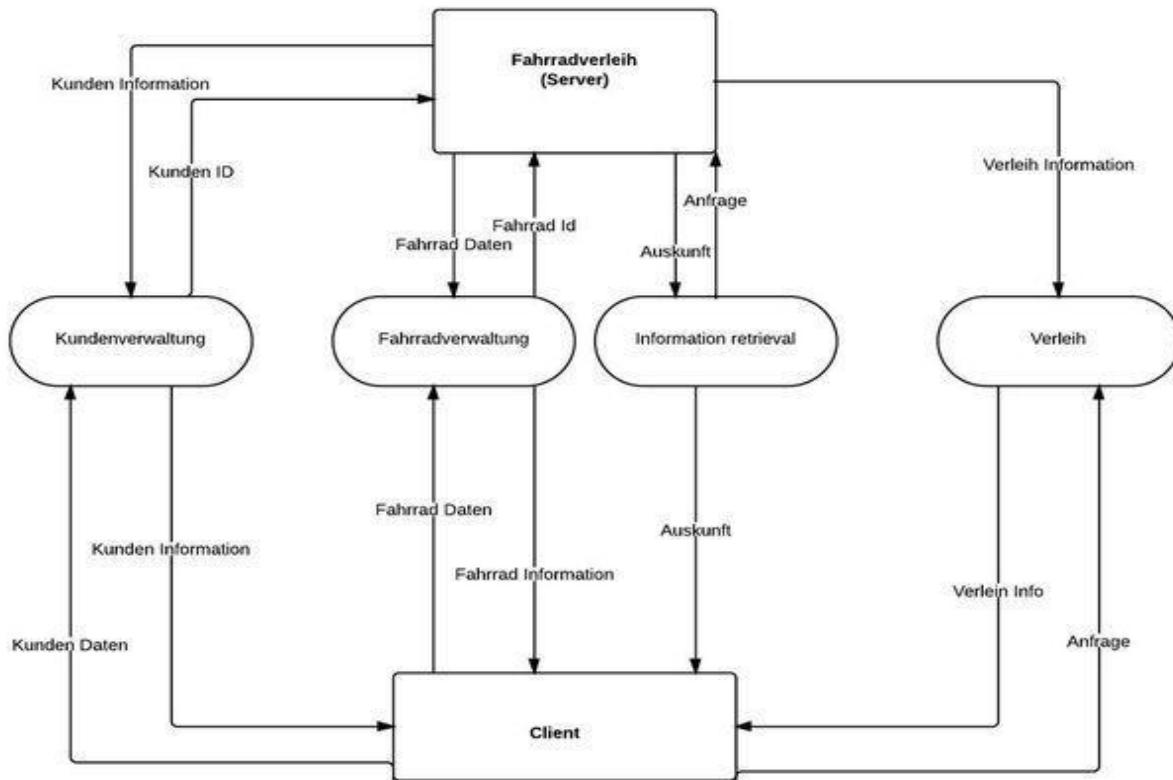
### **Beispielhafter Anwendungsfall**

Peter (User1) hat ein Fahrrad, das er nur gelegentlich braucht. Dieses möchte er anderen Personen, die ein Fahrrad brauchen zur Verfügung stellen und dazu trägt er es mit Herstellername, Model Typ, Rahmen Nummer und dem Betrag den er für das Ausleihen verlangt im Verleihsystem ein.

Hans möchte umweltbewusster von einem Ort zum anderen kommen. Deshalb möchte er sich ein Fahrrad anschaffen.

Er sucht im Verleihsystem nach einem verfügbaren Fahrrad in seiner Nähe. Der Server sucht in seinen Daten nach einem Teil mit dem Typ Fahrrad und übermittelt die unter der id des Anbieters abgelegt IP an Hans, der sich das Fahrrad von Peter aussucht. Nachdem Hans das Fahrrad von Peter abgeholt hat gibt Peter nun im Verleihsystem an, dass sein Fahrrad an Hans vergeben ist. Das System erhält dabei die id von Hans, die bei der Ortsanfrage mit übermittelt wurde, als Ausleihenden.

### Informationsflussdiagramm



---

Dieses Diagramm vermittelt den Datenaustausch zwischen dem Server und dem Client in Bezug auf das geplante System. Doch welche Daten werden vom Server an den Client geschickt und welche vom Client an den Server?.

Der Server verarbeitet sämtliche Kundeninformationen bzw. User-Informationen, Fahrraddaten, Suchanfragen (im Diagramm unter "Auskunft") und die Verleihinformationen. Diese Daten werden anschließend bei einer Anfrage an den Client geleitet.

Das Diagramm beschreibt die Kommunikationsabläufe der besagten Daten.

Anhand dieses Diagramms geht jedoch nicht eindeutig hervor welche Daten vom Server verarbeitet werden und welche Aufgabe der Client erfüllt.



### Registrierung



### Daten bearbeiten (Benutzer- bzw Fahrradd Daten)



### Ausleihen bzw. Reservieren

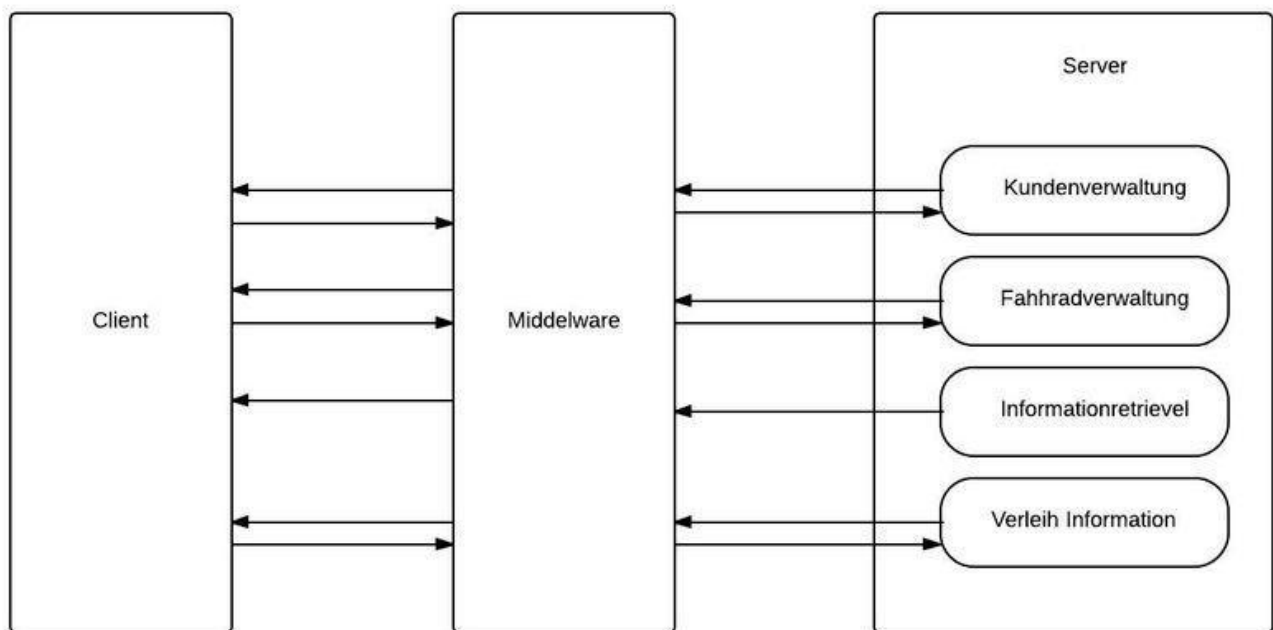


---

Durch einen erneuten Versuch den Datenaustausch zwischen Server und Client darzustellen, entstand das obige Diagramm. Hier sieht man die Hauptfunktionen des Systems, aufgeteilt in Modelle, die den Datenfluss zwischen den beiden Komponenten genauer aufzeigen. Dies war problematisch, da die Schnittstelle zwischen diesen Komponenten fehlte.

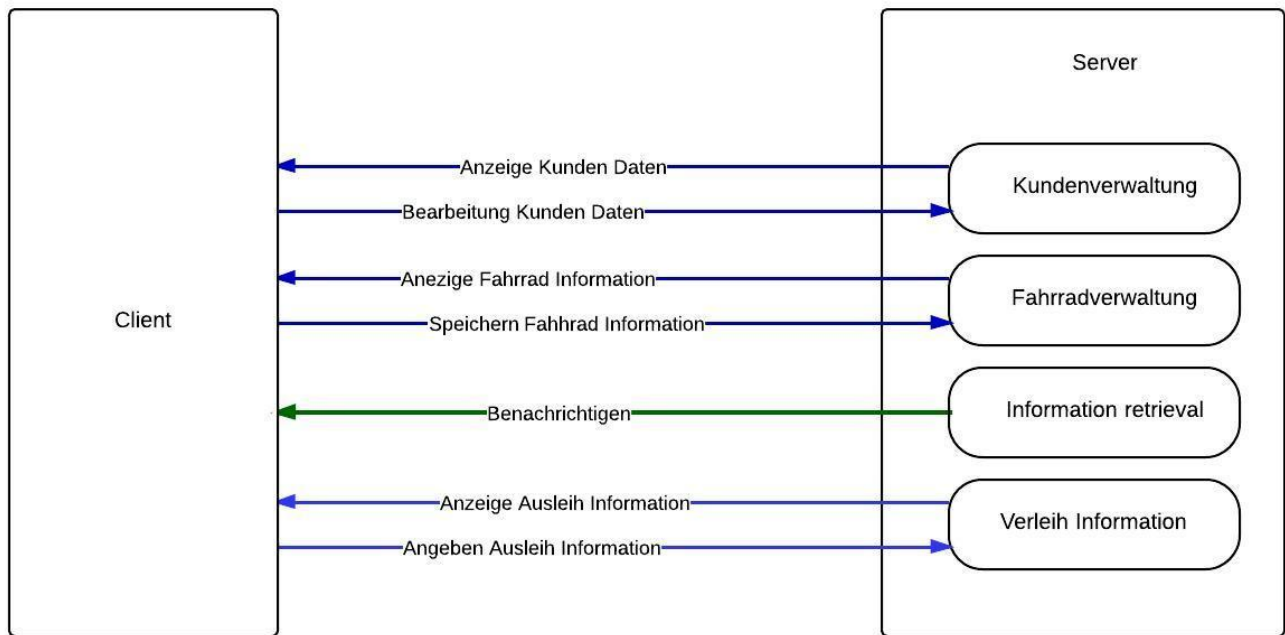
Eine Middleware würde das Schnittstellenproblem beheben. Sie dient für den Transport komplexer Daten und vermittelt u.a. Funktionsaufrufe zwischen den Komponenten.

Informationsflussdiagramm



Letztendlich wurde die Middleware aus der Grafik entfernt, weil es keinen konkreten Zweck in diesem Zusammenhang erfüllt. Zusätzlich wurden die synchronen und asynchronen Kommunikationswege hinzugefügt.

### Informationsflussdiagramm



Synchrone Nachricht →  
Asynchrone Nachricht →

Im weiteren Verlauf sollte die Planung und Konzeption für das Projekt nötiger valider XML Schemata durchgeführt werden. Um es konzeptionell anschaulicher zu machen wird vorerst ein XML-Dokument erstellt.

---

## XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns="http://www.example.org/FahrradVerleih">
  <users>
    <user id="b101">
      <name>muster</name>
      <vorname>kely</vorname>
      <adresse>
        <straße>kely_str</straße>
        <hausNr>5</hausNr>
        <plz>3189</plz>
        <ort>köln</ort>
      </adresse>
      <passwort>12ABcd45</passwort>
    </user>
  </users>
  <fahrraeder>
    <fahrrad id="a202">
      <bild>http://tempuri.org</bild>
      <status>true</status>
      <standort></standort>
      <hersteller>bmw</hersteller>
      <rahmenNr>1232</rahmenNr>
    </fahrrad>
  </fahrraeder>
  <verleih>
    <suche>
      <standort></standort>
    </suche>
    <ausleih fahrradID="a202">
      <rueckgabeort></rueckgabeort>
      <standort></standort>
      <mietBeginn>2001-12-31T12:00:00</mietBeginn>
      <mietEnde>2001-12-31T12:00:00</mietEnde>
    </ausleih>
    <rueckgabe fahrradID="a202">
      <mietEnde>2001-12-31T12:00:00</mietEnde>
      <rueckgabeort></rueckgabeort>
    </rueckgabe>
  </verleih>
</collection>
```

Das Root-Element ist die Collection. Es besteht aus Users, Fahrräder und Verleih. Die Benutzer werden in Users zusammengefasst, wobei der einzelne Benutzer weitere Informationen wie die ID, den Namen, Vornamen, die Adresse und das Passwort

benötigt. Die Adresse besteht wiederum aus Straße, Hausnummer, PLZ und Ort. Diese werden den Elementen durch Attribute mitgegeben.

Das Element Fahrräder benötigt die Attribute ID, Bild, Status, Standort, Hersteller und Rahmennummer.

Außerdem werden für Verleih die Suche mit dem Standort, die Ausleihdaten mit der Fahrrad ID, dem Rückgabeort, dem Standort, dem Mietbeginn, dem Mietende und die Rückgabe mit der Fahrrad ID, dem Mietende und dem Rückgabeort festgelegt.

Jedoch war dieses XML-Dokument unvollständig. Somit wurde folgendes erstellt, wobei die Fahrräder eindeutig dem Benutzer und somit auch einem Besitzer zugeordnet wurden. Zusätzlich wurde der Preis für das jeweilige Fahrrad hinzugefügt und die ID der jeweiligen Elemente als ein Integerwert festgelegt.

### Überarbeitetes XML-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:collection xmlns:tns="http://www.example.org/FahrradVerleih"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/FahrradVerleih FahrradVerleih.xsd">
  <tns:users>
    <tns:user BesitzerVonfahrradID="505" id="1">
      <tns:name>name</tns:name>
      <tns:vorname>vornamej</tns:vorname>
      <tns:adresse>
        <tns:straße>straße</tns:straße>
        <tns:hausNr>5</tns:hausNr>
        <tns:plz>51980</tns:plz>
        <tns:ort>koeln</tns:ort>
      </tns:adresse>
      <tns:passwort>239Om3Ns</tns:passwort>
    </tns:user>

    <tns:user BesitzerVonfahrradID="7" id="2">
      <tns:name>eman</tns:name>
      <tns:vorname>jooti</tns:vorname>
      <tns:adresse>
        <tns:straße>jootistraße</tns:straße>
        <tns:hausNr>35</tns:hausNr>
        <tns:plz>99980</tns:plz>
        <tns:ort>koeln</tns:ort>
      </tns:adresse>
      <tns:passwort></tns:passwort>
    </tns:user>

    <tns:user BesitzerVonfahrradID="8" id="3">
      <tns:name>jemande</tns:name>
      <tns:vorname>etwas</tns:vorname>
      <tns:adresse>
        <tns:straße>egal</tns:straße>
        <tns:hausNr>50</tns:hausNr>
        <tns:plz>89980</tns:plz>
      </tns:adresse>
    </tns:user>
  </tns:users>
</tns:collection>
```

```

    <tns:ort>koeln</tns:ort>
  </tns:adresse>
  <tns:passwort></tns:passwort>
</tns:user>

</tns:users>

<tns:fahrraeder>
  <tns:fahrrad BesitzerID="1" id="505" preis="10.0">
    <tns:bild>http://tempuri.org</tns:bild>
    <tns:status>true</tns:status>
    <tns:standort>50.513427,138.105462</tns:standort>
    <tns:hersteller>BMW</tns:hersteller>
    <tns:rahmenNr>1234</tns:rahmenNr>
    <tns:model>KinderFahrraeder</tns:model>
  </tns:fahrrad>

  <tns:fahrrad BesitzerID="1" VergebenAnID="3" id="606" preis="5.0">
    <tns:bild>http://tempuri.org</tns:bild>
    <tns:status>true</tns:status>
    <tns:standort>50.513427,138.105444</tns:standort>
    <tns:hersteller>BMW</tns:hersteller>
    <tns:rahmenNr>5678</tns:rahmenNr>
    <tns:model>BMX</tns:model>
  </tns:fahrrad>

  <tns:fahrrad BesitzerID="2" id="707" preis="5.0">
    <tns:bild>http://tempuri.org</tns:bild>
    <tns:status>true</tns:status>
    <tns:standort>50.513427,138.105462</tns:standort>
    <tns:hersteller>BMW</tns:hersteller>
    <tns:rahmenNr>9876</tns:rahmenNr>
    <tns:model>Citybike</tns:model>
  </tns:fahrrad>
</tns:fahrraeder>

<tns:verleih>
  <tns:suche>
    <tns:standort>50.513427,138.105462</tns:standort>
    <tns:model>Mountainbike</tns:model>
  </tns:suche>
  <tns:ausleih VergebenAnID="3" fahrradID="606">
    <tns:rueckgabeort>50.513427,138.105462</tns:rueckgabeort>
    <tns:standort>50.513427,138.105462</tns:standort>
    <tns:mietBeginn>2001-12-31T12:00:00</tns:mietBeginn>
    <tns:mietEnde>2001-12-31T12:00:00</tns:mietEnde>
  </tns:ausleih>
  <tns:rueckgabe VergebenAnID="3" fahrradID="707">
    <tns:mietEnde>2001-12-31T12:00:00</tns:mietEnde>
    <tns:rueckgabeort>50.513427,138.105462</tns:rueckgabeort>
  </tns:rueckgabe>
</tns:verleih>
</tns:collection>

```

---

## XML-Schema

Nachdem man ein XML-Dokument erstellt hat, ist es ersichtlich welche Elemente und Attribute für das XML-Schema erforderlich sind. Es gibt mehrere Formen ein Schema zu konstruieren. Die häufigsten sind "Russian Doll Design", "Salami Slice Design" und das "Venetian Blind Design".

Für den ersten Entwurf, der hier wegen der Übersichtlichkeit nicht aufgeführt wird, wurde der "Salami Slice Design" verwendet. Dieses Design hat die Besonderheit, dass das Schema wenig geschachtelt ist und mit Referenzen auf global deklarierte Elemente und Attribute zugreift.

Allerdings war Venetian Blind Design die sinnvollste Variante ein einziges globales Element zu definieren, die viele globale Typen enthält. Dadurch wird garantiert, dass die Struktur übersichtlich und leicht zu verstehen ist. Das Rootelement ist eindeutig und man gewinnt an Wiederverwendung da alle wichtigen Typen global deklariert werden können. Diese Vorteile bringt das "Salami Slice Design" und das "Russian Doll Design" nicht. Zudem können in dem XML-Dokument nicht nur die Elemente sondern auch die Attribute ohne Namespace-Spezifizierung verwendet werden.

## XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/FahrradVerleih"
xmlns:tns="http://www.example.org/FahrradVerleih" attributeFormDefault="unqualified"
elementFormDefault="qualified">

  <!-- Börse "Kollektion -->
  <element name="collection">
    <complexType>
      <sequence>
        <element name="users" type="tns:users_type"/></element>
        <element name="fahrraeder" type="tns:fahrraeder_type"/></element>
        <element name="verleih" type="tns:verleih_type"/></element>
      </sequence>
    </complexType>
  </element>

  <!-- Users -->
  <complexType name="users_type">
    <sequence>
      <element name="user" type="tns:user_type" maxOccurs="unbounded"/></element>
    </sequence>
  </complexType>
```

```

<!-- Passwort Beschränkung "soll 8 Zeichen lang sein und kann a-z , A-Z
und 0-9 beinhalten " -->

<simpleType name="passwort_type">
  <restriction base="string">
    <pattern value="[a-zA-Z0-9]{8}" />
  </restriction>
</simpleType>

<!-- User Daten -->
<complexType name="user_type">
  <sequence>
    <element name="name" type="string"></element>
    <element name="vorname" type="string"></element>
    <element name="adresse" type="tns:adresse_type"></element>
    <element name="passwort" type="tns:passwort_type"></element>

    <!-- <element name="besitzerVon" type="tns:besitzerVon_type"> </element> -->
  </sequence>
  <attribute name="id" type="integer" use="required"></attribute>
  <attribute name="BesitzerVonfahrradID" type="integer" use="required" />
</complexType>

<!-- Adresse Daten -->
<complexType name="adresse_type">
  <sequence>
    <element name="straße" type="string"></element>
    <element name="hausNr" type="integer"></element>
    <element name="plz" type="integer"></element>
    <element name="ort" type="string"></element>
  </sequence>
</complexType>

<!-- fahrraeder Kollektion -->
<complexType name="fahrraeder_type">
  <sequence>
    <element name="fahrrad" type="tns:fahrrad_type"
maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<!-- Fahrrad Daten -->
<complexType name="fahrrad_type">
  <sequence>
    <element name="bild" type="anyURI"></element>
    <element name="status" type="boolean"></element>
    <element name="standort" type="tns:koord_type"></element>
    <element name="hersteller" type="string"></element>
    <element name="rahmenNr" type="integer"></element>
  </sequence>
</complexType>

```



```

        <element name="model" type="tns:model_type"></element>
    </sequence>
    <attribute name="id" type="integer" use="required"></attribute>
    <attribute name="BesitzerID" type="integer" use="required" />
    <attribute name="VergebenAnID" type="integer" />
    <attribute name="preis" type="float"></attribute>

</complexType>

<!-- Fahrrad Modelle -->
<simpleType name="model_type">
    <restriction base="string">
        <enumeration value="Mountainbike" />
        <enumeration value="Rennraeder" />
        <enumeration value="BMX" />
        <enumeration value="Citybike" />
        <enumeration value="KinderFahrraeder" />
    </restriction>
</simpleType>

<!-- Standort and rueckgabeort Koordinaten -->
<simpleType name="koord_type">
    <restriction base="string">
        <pattern
            value="-?([0-9][1-9][0-9]1[0-7][0-9]).[0-9]{4,6},-?([0-9][1-9][0-9][1-2][0-9][0-9]3[0-5][0-9]).[0-9]{4,6}" />
    </restriction>
</simpleType>

<!-- Verleih Information -->
<complexType name="verleih_type">
    <sequence>
        <element name="suche" type="tns:suche_type"></element>
        <element name="ausleih" type="tns:ausleih_type"></element>
        <element name="rueckgabe" type="tns:rueckgabe_type"></element>
    </sequence>
</complexType>

<!-- Fahrrad Suchkriterien -->
<complexType name="suche_type">
    <sequence>
        <element name="standort" type="tns:koord_type"></element>
        <element name="model" type="tns:model_type"></element>
    </sequence>
</complexType>

<!-- Ausleih Infomation -->
<complexType name="ausleih_type">
    <sequence>

```

```

        <element name="rueckgabeort" type="tns:koord_type"></element>
        <element name="standort" type="tns:koord_type"></element>
        <element name="mietBeginn" type="dateTime"></element>
        <element name="mietEnde" type="dateTime"></element>
    </sequence>
    <attribute name="VergebenAnID" type="integer" use="required" />
    <attribute name="fahrradID" type="integer" use="required" />
</complexType>

<!-- rueckgabe Information -->
<complexType name="rueckgabe_type">
    <sequence>
        <element name="mietEnde" type="dateTime"></element>
        <element name="rueckgabeort" type="tns:koord_type"></element>
    </sequence>
    <attribute name="VergebenAnID" type="integer" use="required" />
    <attribute name="fahrradID" type="integer" use="required" />
</complexType>

</schema>

```

Wir fassen die gesamten Daten im Root-Element Collection zusammen, darunter gibt es die Elemente Users und Fahrräder und Verleih, die auch die einzelnen User, Fahrräder und den Verleih zusammenfassen. Diese werden dann wiederum durch das Attribut id weiter definiert, im Falle der User werden diese damit eindeutig dem registrierten Nutzern des

Systems zugeordnet, wodurch die persönlichen Daten nicht an verschiedenen Stellen redundant gespeichert werden müssen. In Fahrrad\_Type werden die Daten über das Fahrrad gespeichert (wie: Bild, Status, Standort, Hersteller, Rahmennummer, Modell sowie der Besitzer und an wen das Fahrrad verliehen ist). Koor\_type beschreibt den Rückgabeort und den Standort des Fahrrades. Dies wird anhand der aktuellen Koordinaten festgelegt. Unter den Verleih werden alle Vorgänge gespeichert, die für den Verleihvorgang nötig sind z.B. die Suche, die Ausleihe und die Rückgabe. Die Suche verwendet den Standort anhand der Koordinaten und das Modell des Fahrrades. In der Ausleihe werden der Standort, Rückgabeort, der Zeitraum (Mietbeginn und Mietende), sowie die User Daten des Mieters und die FahrradID gespeichert. Um deutlich zu machen ob ein Fahrrad zurück gegeben wurde, werden die Daten Mietende, der Rückgabeort, ID des Mieters und des Fahrrades gespeichert.

### Kritik und Verbesserung:

Trotz der detaillierten Vorgehensweise, die hier angewendet wurde, kam es zu Unstimmigkeiten in Bezug auf die Daten und die Referenzierung sowie den Redundanzen. Die Verleihdaten stellen das Problem dar, dass sie ohne das ein Fahrrad gespeichert sind nicht existieren konnten. Deshalb wurde der Verleih aus der Collection

---

entnommen. Desweiteren sind Redundanzen in Bezug auf den AusleihType vorhanden. Die FahrradID und VergebenAnID sowie Standort wurden aus dem AusleihType entfernt, da sie bereits in FahrradType vorhanden sind. Zudem wurde der Preis zu FahrradType hinzugefügt.

Außerdem ist SucheType überflüssig. Die Suche kann als Query definiert werden und somit ist es sinnvoll dies aus dem XML-Schema zu entfernen.

Das neue XML-Schema ist nach der Verbesserung wesentlich kompakter und es treten keine Redundanzen mehr auf. Die Abhängigkeit der einzelnen Elemente ist somit klar definiert.

## Überarbeitetes XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/FahrradVerleih"
xmlns:tns="http://www.example.org/FahrradVerleih" attributeFormDefault="unqualified"
elementFormDefault="qualified">

  <!-- Börse "Kollektion" -->
  <element name="collection">
    <complexType>
      <sequence>
        <element name="users" type="tns:users_type"/>
        <element name="fahrtraeder" type="tns:fahrtraeder_type"/>
      </sequence>
    </complexType>
  </element>

  <!-- Users -->
  <complexType name="users_type">
    <sequence>
      <element name="user" type="tns:user_type" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <!-- User Daten -->
  <complexType name="user_type">
    <sequence>
      <element name="name" type="string"/>
      <element name="vorname" type="string"/>
      <element name="passwort" type="string"/>
      <element name="strasse" type="string"/>
      <element name="hausNr" type="integer"/>
      <element name="plz" type="integer"/>
      <element name="ort" type="string"/>
    </sequence>
    <attribute name="id" type="integer" use="required"/>
  </complexType>
```

```

<!-- fahrraeder Kollektion -->
<complexType name="fahrraeder_type">
  <sequence>
    <element name="fahrrad" type="tns:fahrrad_type"
maxOccurs="unbounded"/></element>
  </sequence>
</complexType>

<!-- Fahrrad Daten -->
<complexType name="fahrrad_type">
  <sequence>
    <element name="bild" type="anyURI"/></element>
    <element name="standort" type="tns:koord_type"/></element>
    <element name="hersteller" type="string"/></element>
    <element name="rahmenNr" type="integer"/></element>
    <element name="model" type="tns:model_type"/></element>
    <element name="vergebenAn" type="tns:vergebenAn_type"/></element>
    <element name="ausleih" type="tns:ausleih_type"/></element>
  </sequence>
  <attribute name="id" type="integer" use="required"/></attribute>
  <attribute name="BesitzerID" type="integer" use="required" />
  <attribute name="preis" type="float"/></attribute>
</complexType>

<complexType name="vergebenAn_type">
  <attribute name="id" type="integer"/></attribute>
</complexType>

<!-- Fahrrad Modelle -->
<simpleType name="model_type">
  <restriction base="string">
    <enumeration value="Mountainbike" />
    <enumeration value="Rennraeder" />
    <enumeration value="BMX" />
    <enumeration value="Citybike" />
    <enumeration value="KinderFahrraeder" />
  </restriction>
</simpleType>

<!-- Standort and rueckgabeort Koordinaten -->
<simpleType name="koord_type">
  <restriction base="string">
    <pattern
value="-?([0-9][1-9][0-9]1[0-7][0-9]).[0-9]{4,6},-?([0-9][1-9][0-9][1-2][0-9][0-9]3[0-5][0-9]).[0-9]{4,6}" />
    </pattern>
  </restriction>
</simpleType>

```

---

```
<!-- Ausleih Infomation -->
<complexType name="ausleih_type">
  <sequence>
    <element name="rueckgabeort" type="tns:koord_type"/></element>
    <element name="mietBeginn" type="dateTime"/></element>
    <element name="mietEnde" type="dateTime"/></element>
  </sequence>
</complexType>
</schema>
```

---

## Ressourcen und die Semantik der HTTP-Operationen

Um auf die Ressourcen zuzugreifen müssen diese vorerst definiert werden. Folgende Punkte müssen bei der Definition beachtet werden:

**“Adressierbarkeit:** Jede Ressource muss über einen eindeutigen ***Unique Resource Identifier*** (kurz URI) identifiziert werden können. Ein Kunde mit der Kundennummer 123456 könnte also zum Beispiel über die URI `http://ws.mydomain.tld/customers/123456` adressiert werden.

**Zustandslosigkeit:** Die Kommunikation der Teilnehmer untereinander ist zustandslos. Dies bedeutet, dass **keine Benutzersitzungen** (etwa in Form von Sessions und Cookies) existieren, sondern bei jeder Anfrage alle notwendigen Informationen wieder neu mitgeschickt werden müssen.

Durch die Zustandslosigkeit sind REST-Services sehr einfach skalierbar; da keine Sitzungen existieren, ist es im Grunde egal, wenn mehrere Anfragen eines Clients auf verschiedene Server verteilt werden.

**Einheitliche Schnittstelle:** Jede Ressource muss über einen einheitlichen Satz von Standardmethoden zugegriffen werden können. Beispiele für solche Methoden sind die Standard-HTTP-Methoden wie GET, POST, PUT, und mehr.

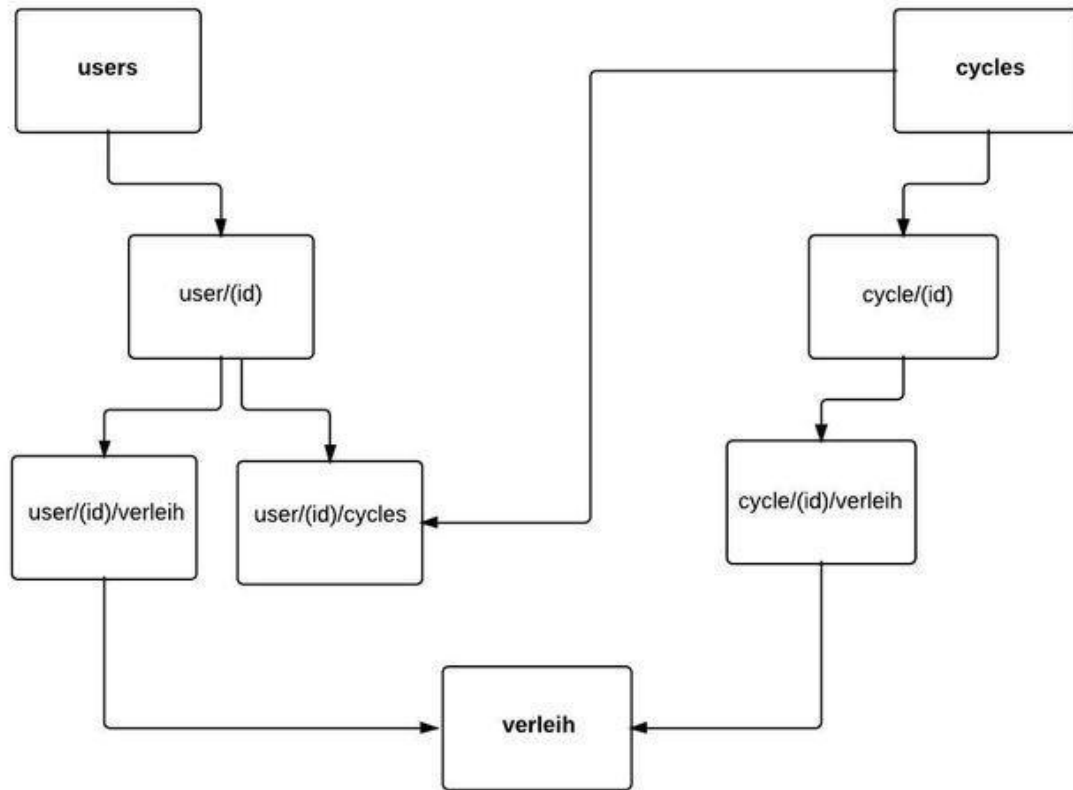
**Entkopplung von Ressourcen und Repräsentation:** Das bedeutet, dass verschiedene Repräsentationen einer Ressource existieren können. Ein Client kann somit etwa eine Ressource explizit beispielsweise im XML- oder JSON-Format anfordern.”<sup>1</sup>

Es wurden zwei (Ober-) Ressourcen definiert: Users, unter dem die User abgelegt werden und Fahrräder, die ein Fahrrad beinhaltet. Wie es in dem obigen XML-Schema festgelegt ist, kann man über diese zwei Ressourcen auf die wichtigsten Funktionen des Systems zugreifen. Sie sind voneinander unabhängig.

---

<sup>1</sup> <https://blog.mittwald.de/webentwicklung/restful-webservices-1-was-ist-das-uberhaupt>

## Ressourcen



Bei Ressourcen handelt es sich um Objekte, die Identifizierbar sind und über eine oder mehrere Repräsentationen verfügen. Somit sind sie für Nutzer erreichbar und können bearbeitet werden.

Ressourcen bilden das zentrale Konzept von REST. Folgende Methoden wurden vorerst für die Anwendung in Betracht bezogen.

Ressource	URI	Methode
Liste aller Fahrräder	/fahrraeder/	GET, POST  Query: userid(Besitzer) rahmenNr model verfügbar standort

---

Einzelne Fahrrad	/fahrrader/fahrrad/(id)	GET, PUT, POST, DELETE
Liste aller User	/users/	GET, POST
Einzelne User	/user/user/(id)	GET,PUT,POST,DELETE
Verleih information	/vergebenAn/	GET,PUT,DELETE
Liste aller Fahrräder des Benutzers	/users/user/(id)/fahrraeder	GET,POST Query: fahrradid model hersteller rahmenNr vergebenAn (vergebene Fahrraeder) standort ausleihInfo
Gibt vergebenAn eines bestimmten Fahrrad des Verleihsystems aus	/fahrraeder/fahrrad/(id)/vergebenAn	GET,PUT,POST,DELETE
Verleihinformation der Fahrräder	/fahrraeder/fahrrad/(id)/verleih	GET,POST

GET= Lesezugriff auf Ressourcen → Daten dürfen auf dem Server nicht verändert werden

POST= Erstellen von neuen Ressourcen → sind noch nicht vorhanden, neue URI

PUT= Erstellen und bearbeiten von Ressourcen → URI bereits bekannt

DELETE= Löschen von Ressourcen



---

Da das XML-Schema im späteren Verlauf angepasst wurde, wird Ausleihe als Unterressource von Fahrrädern dargestellt.

Ressource	URI	Methode
Liste aller Fahrräder	/fahrraeder/	GET, POST  Query: userid(Besitzer) rahmenNr model verfügbar standort
Einzelne Fahrrad	/fahrraeder/fahrrad/(id)	GET, PUT, POST, DELETE
Liste aller User	/users/	GET, POST
Einzelne User	/user/user/(id)	GET,PUT,POST,DELETE
VergebenAn eines Fahrrad.	/fahrraeder/fahrrad/(fahrradid)/vergebenAn	GET,PUT,POST,DELETE
Liste aller Fahrräder des Benutzers	/users/user/(id)/fahrraeder	GET,POST  Query: fahrradid model hersteller rahmenNr vergebenAn
AusleihInformation eines bestimmten Fahrrad	/fahrraeder/fahrrad/(fahrradid)/ausleih	GET,PUT,POST,DELETE

---

## RESTful Webservice

Als Ausgangspunkt dienen die mit JAXB erstellten Java-Klassen der Datenstruktur. Erforderliche Methoden wurden bereits hinzugefügt. Dabei handelt es sich hauptsächlich um getter- und setter- Methoden die die Klasse FahrradService (package: minirestwebservice) benötigt. Diese sowie FahrradServer und FahrradClient wurden anhand der verlinkten Dokumentationen und Codebeispielen selbst erstellt.

Die Service Klasse enthält den tatsächlichen Web Service, dort sind die definierten Operationen für die Ressourcen implementiert. Der eigentliche (Konsolenbasierte-) Client befindet sich im package de.fahrrad.menu. Dort erzeugt FahrradMenu die Konsolen Ein und Ausgabe, FahrradRequest führt die jeweiligen Operationen (GET, PUT, POST, DELETE) mit den eingegebenen Inhalten, die vom Service Serverseitig bearbeitet werden, aus.

Die XMLOutputFormat (<http://stackoverflow.com/a/1264912>) Klasse ist hingegen nicht selbstgeschrieben und übernimmt die Umwandlung des String, der von der GET Methode erzeugt wird, in eine Formatierte XML Ausgabe.

### Methoden:

getAll(): Gibt den kompletten Inhalt des Verleihsystems aus.

gerUsers(): Gibt alle eingetragenen User des Verleihsystems aus.

postUsers(): Erzeugt einen neuen User im Verleihsystem.

getOneUser(): Gibt einen bestimmten eingetragenen User des Verleihsystems aus.

putOneUser(): Ändert Eigenschaften eines bestimmten Users des Verleihsystems

postOneUser(): Erzeugt ein neues Fahrrad zu einem bestimmten User des Verleihsystems (= postFahrraeder) aber unterscheidet sich bei Path( "/users/user/{userid}")

deleteOneUser(): Löschen eines bestimmten Users im Verleihsystems.

getOneUserFahrraeder(): Gibt alle Fahrräer eines bestimmten Users des Verleihsystems aus.

postOneUserFahrraeder(): Erzeugt ein neues Fahrrad für einen bestimmten User des Verleihsystems. (= postFahrraeder) Path( "/users/user/{userid}/fahrraeder")

getFahrraeder(): Gibt alle Fahrrads des Verleihsystems entsprechend der Suchkriterien aus.

---

postFahrraeder(): Erzeugt ein neues Fahrrad für einen bestimmten User des Verleihsystems. Path( "/fahrraeder")

getOneFahrrad(): Gibt ein bestimmtes Fahrrad des Verleihsystems aus.

putOneFahrrad(): Ändert Eigenschaften eines bestimmten Users des Verleihsystems.

deleteOneFahrrad(): Löschen eines bestimmten Fahrrades im Verleihsystem.

getVergebenAn(): Gibt vergebenAn eines bestimmten Fahrrades im Verleihsystem aus.

postVergebeebAn(): Erzeugt ein vergebenAn für ein bestimmtes Fahrrad des Verleihsystems.

putVergebenAn(): Ändert vergebenAn eines bestimmten Fahrrades des Verleihsystems.

deleteVergebenAn(): Löscht vergebenAn eines bestimmten Fahrrades des Verleihsystems.

getAusleih(): Gibt AusleihInformation eines bestimmten Fahrrades des Verleihsystems aus.

postAusleih(): Erzeugt eine Ausleihe eines bestimmten Fahrrad des Verleihsystems.

putAusleih(): Ändert Ausleihe eines bestimmten Fahrrades des Verleihsystems.

deleteAusleih(): Löscht AusleihInformation eines bestimmten Fahrrades des Verleihsystems.

---

## Operationen:

### GET /users/

#### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼<collection xmlns="http://www.example.org/FahrradVerleih">
  ▼<users>
    ▶<user id="1">...</user>
    ▶<user id="2">...</user>
    ▶<user id="3">...</user>
    ▶<user id="4">...</user>
  </users>
</collection>
```

### Path: /users/

GET: Liste aller User

PUT: -

POST: User hinzufügen

DELETE: -

### GET /users/user/userid

#### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼<collection xmlns="http://www.example.org/FahrradVerleih">
  ▼<users>
    ▼<user id="1">
      <name>zuckerberg</name>
      <vorname>mark</vorname>
      <password>smackdoesntwork</password>
      <strasse>irgendwo</strasse>
      <hausNr>5</hausNr>
      <plz>267</plz>
      <ort>london</ort>
    </user>
  </users>
</collection>
```

### Path: /users/user/{userid}

GET: Informationen zu dem User mit der ID {id}

---

---

PUT: Ändern des Benutzers mit der ID {id}

POST: Zusätzliches Fahrrad hinzufügen, dass der User {id} anbieten möchte

DELETE: User {id} löschen

## GET /users/user/userid/fahrraeder

### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼<collection xmlns="http://www.example.org/FahrradVerleih">
  ▼<fahrraeder>
    ▶<fahrrad id="100" BesitzerID="1" preis="2.0">...</fahrrad>
    ▶<fahrrad id="101" BesitzerID="1" preis="5.0">...</fahrrad>
    ▶<fahrrad id="105" BesitzerID="1" preis="13.0">...</fahrrad>
  </fahrraeder>
</collection>
```

## Path: /users/user/userid/fahrraeder

GET: Liste aller Fahrräder des Benutzers {id}

PUT: -

POST: Fahrrad hinzufügen, der Besitzer ist {id}

DELETE: -

## GET /fahrraeder

### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼<collection xmlns="http://www.example.org/FahrradVerleih">
  ▼<fahrraeder>
    ▶<fahrrad id="100" BesitzerID="1" preis="2.0">...</fahrrad>
    ▶<fahrrad id="101" BesitzerID="1" preis="5.0">...</fahrrad>
    ▶<fahrrad id="102" BesitzerID="2" preis="5.0">...</fahrrad>
    ▶<fahrrad id="103" BesitzerID="2" preis="6.0">...</fahrrad>
    ▶<fahrrad id="104" BesitzerID="3" preis="3.0">...</fahrrad>
    ▶<fahrrad id="105" BesitzerID="1" preis="13.0">...</fahrrad>
  </fahrraeder>
</collection>
```

## Path:/fahrraeder

GET: Liste aller Fahrräder

---

---

PUT: -

POST: Fahrrad hinzufügen

DELETE: -

**GET /fahrtraeder?verfuegbar**

**Ausgabe:**

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼<collection xmlns="http://www.example.org/FahrradVerleih">
  ▼<fahrtraeder>
    ▶<fahrrad id="101" BesitzerID="1" preis="5.0">...</fahrrad>
    ▶<fahrrad id="103" BesitzerID="2" preis="6.0">...</fahrrad>
  </fahrtraeder>
</collection>
```

GET: Zum ausleihen verfügbare Fahrräder

(Fahrräder können auch nach Hersteller, Model, Fahrrad-Id und nach Standort abgefragt werden.)

---

## GET /fahrraeder/fahrrad/fahrradid

### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
<?xml version="1.0"?>
<collection xmlns="http://www.example.org/FahrradVerleih">
  <fahrraeder>
    <fahrrad id="101" BesitzerID="1" preis="5.0">
      <bild>http://tempuri.org</bild>
      <standort>50.513427,138.105555</standort>
      <hersteller>hersteller</hersteller>
      <rahmenNr>5678</rahmenNr>
      <model>Citybike</model>
    </fahrrad>
  </fahrraeder>
</collection>
```

## Path: /fahrraeder/fahrrad/fahrradid

GET: Ausführliche Informationen zu dem Fahrrad

PUT: Ändern der Informationen des Fahrrades (Es dürfen keine neuen Artikel erstellt werden und es darf die ID nicht geändert werden!)

POST: VergebenAn setzen

DELETE: Löschen des Fahrrades

## GET /fahrraeder/fahrrad/fahrradid/ausleih

### Ausgabe:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
<?xml version="1.0"?>
<collection xmlns="http://www.example.org/FahrradVerleih">
  <fahrraeder>
    <fahrrad>
      <ausleih>
        <rueckgabeort>50.513427,138.10765</rueckgabeort>
        <mietBeginn>2013-6-31T12:00:00</mietBeginn>
        <mietEnde>2013-7-6T10:50:00</mietEnde>
      </ausleih>
    </fahrrad>
  </fahrraeder>
</collection>
```

## Path: /fahrraeder/fahrrad/fahrradid/ausleih

GET: gibt die ganze Ausleih Information (Rückgabeort, MietBeginn und MietEnde) aus

PUT: Ändert die Mietbeginn, Mietende und Rückgabeort

POST: - Erstellt die Ausleihe

---

---

DELETE: - löscht die ganze Ausleihe

**Probleme:**

Folgende Probleme treten beim Ausführen des FahrradClient auf:

- Fahrrad hinzufügen → "500 Internal Server Error"
- Fahrrad bearbeiten → Daten werden nicht geändert
- Fahrrad ausleihen → "404 Not Found"
- Ausleihe Information wird nicht gespeichert.



---

## Konzeption + XMPP Server einrichten

LeafNodes sind Topics, in denen etwas veröffentlicht wird und somit auch asynchrone Benachrichtigungen. Als Topics werden in dieser Anwendung der Standort definiert. Wenn ein Nutzer benachrichtigt werden möchte, wenn sich ein verfügbares Fahrrad in der Nähe befindet, wird eine Nachricht per E-Mail, SMS oder als Push-Benachrichtigung an den User gesendet. Hierbei wäre es möglich die Typen als Collection Nodes zu definieren und darunter einer verfeinerte Suchmöglichkeit zu bieten z.B über den Hersteller. Dies war für das Konzept nicht erforderlich, da es für den Subscriber wichtiger ist ein Fahrrad eines bestimmten Types auszuleihen. Die Suche über den Hersteller ect wäre optional, deshalb wäre es sinnvoll die Lösung mit den Typen als Nodes umzusetzen. Eine Möglichkeit wäre eine Verknüpfung zwischen

PubSubService und dem RestfulWebService und könnte eine URI innerhalbdes WebServices des entsprechendem Fahrrades an den Subscriber weitergeben, welcher sich mit dieser wiederum das Fahrrad auf dem Webservice genauer anschauen kann. Aus zeitlichen Gründen konnte dies nicht Umgesetzt werden.

### Publisher und Subscriber

Publisher sind in diesem Fall die Nutzer die ein Fahrrad zur Verfügung stellen und dieses ausleihbar ist. Sobald ein Fahrrad in der Nähe ist soll eine Benachrichtigung erstellt werden, die dem Subscriber eine Nachricht sendet. Subscriber sind die Personen, die sich im System befinden und benachrichtigt werden möchten. Wenn dies der Fall ist muss der Standort freigegeben werden, damit dieser Dienst genutzt werden kann.

Ein Nutzer kann sowohl als Publisher auftreten, wenn er ein Fahrrad zu verleihen hat, sowie auch als Subscriber, wenn er ein Fahrrad benötigt. Wenn ein Fahrrad vergeben wurde soll es wieder aus der Queue gelöscht werden, sodass nur verfügbare Fahrräder angezeigt werden. Sinnvollerweise würde bei diesem Verfahren nur eine URI zum entsprechenden Fahrrad übermittelt, möglich wäre hier die Verknüpfung mit dem Webservice.

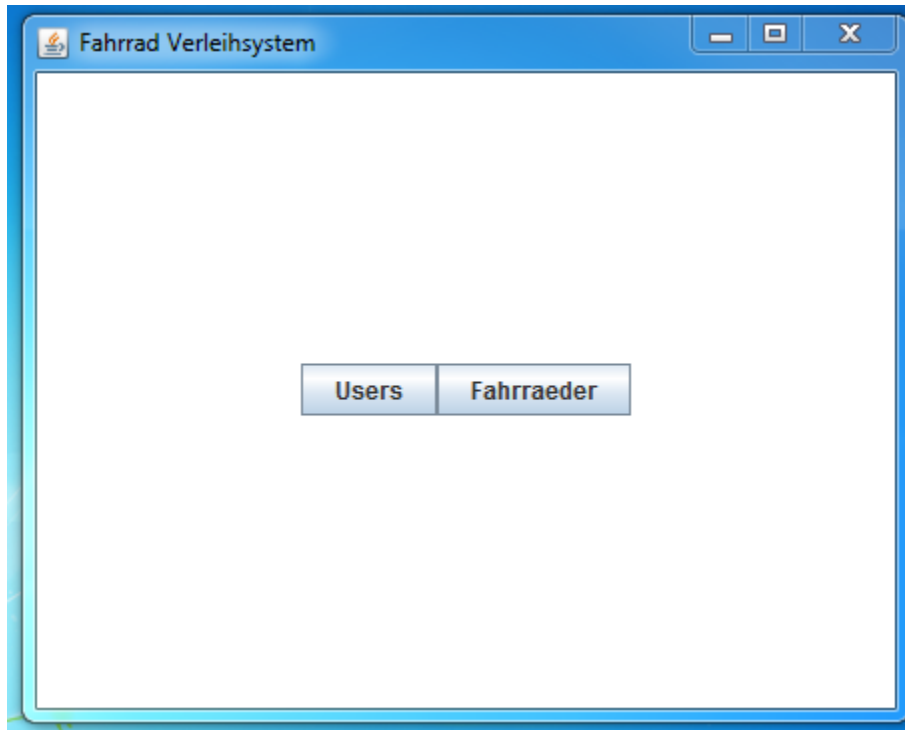
## Client - Entwicklung

Für den Webservice wurde bereits ein Konsolenbasierter Client umgesetzt und die Request-Methoden in einer anderen Klasse definiert, diese werden nun für den Swing-Client wiederverwendet. Der Swing Client wurde weitgehend umgesetzt. Es ist möglich User hinzuzufügen (add), zu ändern (edit) und zu löschen (delete). Die Fahrräder können ebenso aufgeführt werden, jedoch gibt es nicht erklärbare Fehler bei postFahrräder (die schon im FahrradService auftauchten).

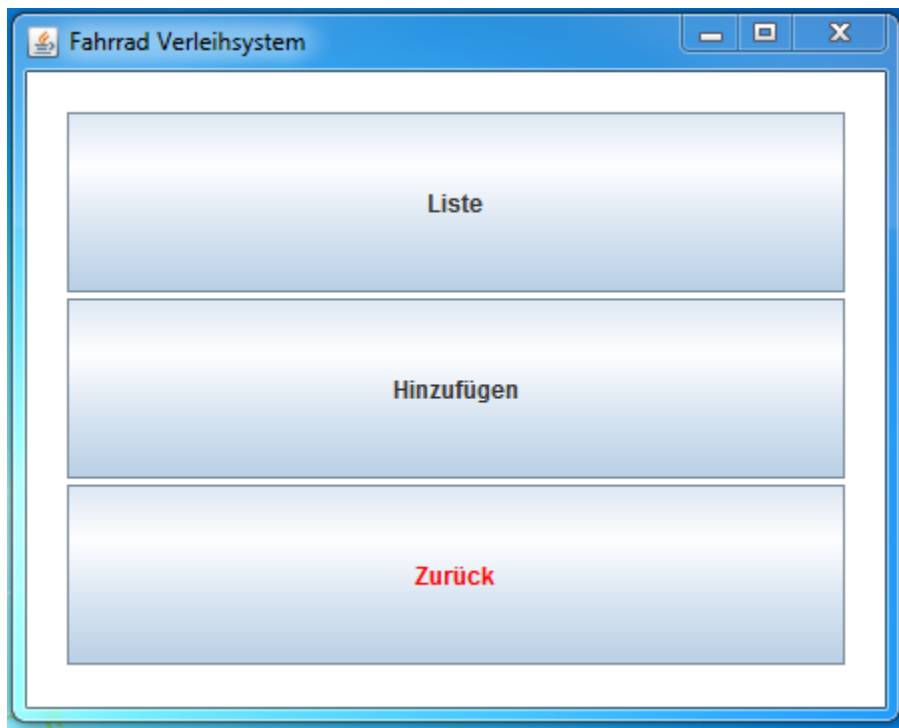
---

## Screenshots

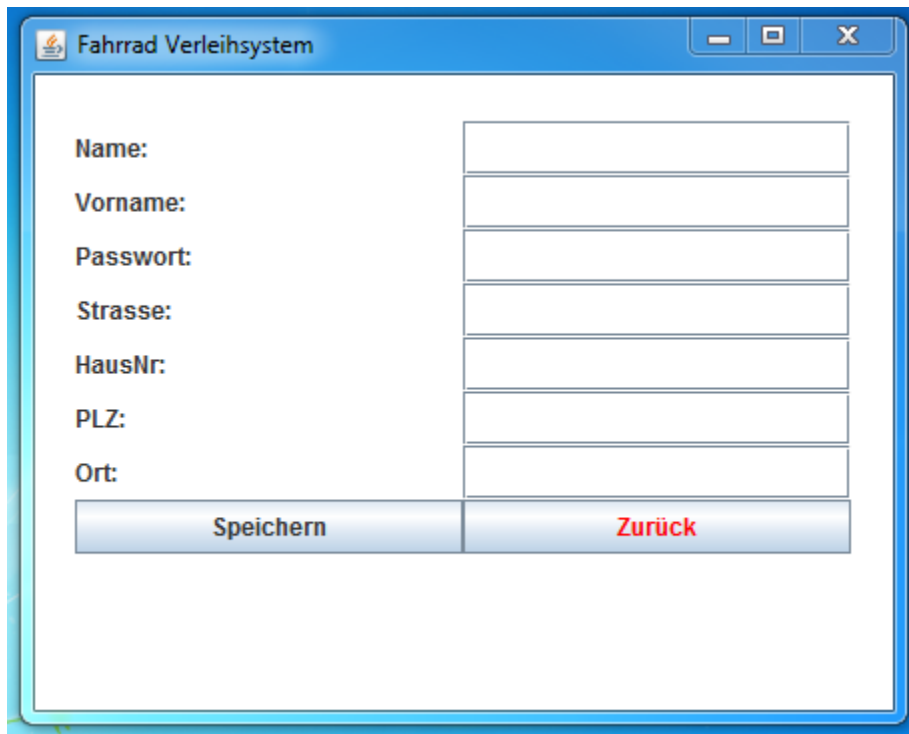
### Das Hauptmenü



### Das Usersmenü



## Das UserAdd-Menü



The screenshot shows a window titled "Fahrrad Verleihsystem" with a standard Windows XP-style title bar. Inside the window, there is a form for adding a new user. The form consists of seven text input fields, each preceded by a label: "Name:", "Vorname:", "Passwort:", "Strasse:", "HausNr:", "PLZ:", and "Ort:". Below these fields are two buttons: "Speichern" (Save) and "Zurück" (Back). The "Zurück" button is highlighted in red.

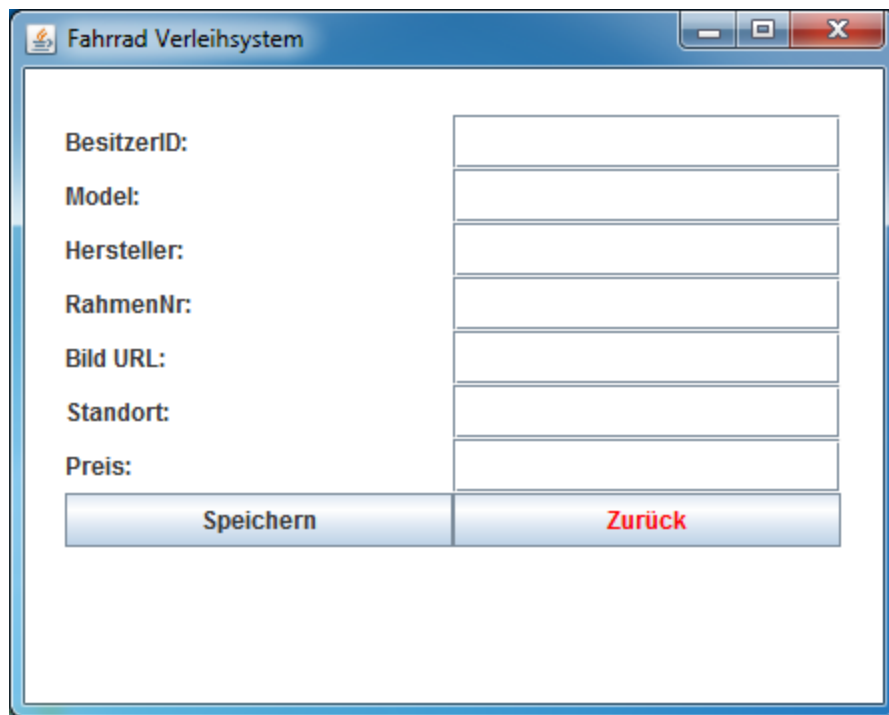
## Das Fahrradmenü



The screenshot shows a window titled "Fahrrad Verleihsystem" with a standard Windows XP-style title bar. Inside the window, there is a menu for managing bicycles. The menu consists of four buttons stacked vertically: "Liste", "Hinzufügen" (Add), "Suchen" (Search), and "Zurück" (Back). The "Zurück" button is highlighted in red.

## Das FahrradAdd-Menü

---



**Fahrrad Verleihsystem**

BesitzerID:

Model:

Hersteller:

RahmenNr:

Bild URL:

Standort:

Preis:

---

## Literaturverzeichnis

### Bücher

- Stefan Tilkov, **REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien**.
- Subbu Allamaraju, **RESTful WEb Services Cookbook**, O'REILLY
- Bill Burke, **RESTful Java: with JAX-RS**, O'REILLY
- Marco Skulschus, Marcus Wiederstein, Sarah Winterstone, **XML Schema**, Comelio
- Peter Saint-Andre, Kevin Smith & Remko Troncon, **XMPP: The Definitive Guide**, O'REILLY
- Helmut Vonhoegen, **Einstieg in XML**, Galileo Press
- Daniel KOCH, **XML für Webentwickler: Ein Praktischer Einstieg**, HANSER

### Internet Links

- [Dokumentation Grizzly Server](#)
- [Codebeispiel mit Jersey und Grizzly](#)
- [Java Swing Tutorial](#)
- [XML Schema Referenzierung](#)
- [XML Schema Design](#)
- [Java Swing](#)