



Bike-Sharing Webservice mit Client-Applikation

# **BIKE-SHARING WEBSERVICE MIT CLIENT-APPLIKATION**

Tobias Hafermalz, Philipp Thöricht  
Mat-Nrn.: , 3485596

# CONTENTS

<b>1 Aufgabe</b>	<b>3</b>
<b>2 Installation</b>	<b>4</b>
2.1 Datenbank . . . . .	4
2.2 Webservice und API . . . . .	4
<b>3 Funktionen Webservice</b>	<b>5</b>
3.1 GET Methoden . . . . .	5
3.1.1 stations . . . . .	5
3.1.2 bikes . . . . .	6
3.1.3 models . . . . .	7
3.1.4 account . . . . .	8
3.1.5 bookings . . . . .	9
3.2 POST Methoden . . . . .	10
3.2.1 bookings . . . . .	10
3.3 DELETE Methoden . . . . .	11
3.3.1 bookings . . . . .	11
3.4 PUT Methoden . . . . .	11
3.4.1 bookings . . . . .	11
<b>4 Webclient</b>	<b>13</b>

<b>5</b>	<b>Zusatzaufgabe WS-Security</b>	<b>15</b>
5.1	OAuth2 . . . . .	15
5.2	Implementierung . . . . .	15
<b>6</b>	<b>Fazit</b>	<b>17</b>
<b>7</b>	<b>Quellen</b>	<b>18</b>

# 1 AUFGABE

Die Aufgabe bestand in einer Implementierung eines Webservices in PHP mit dem Slim Framework. Als Zusatzaufgabe sollte das Thema WS-Security durch Authentifizierung mittels OAuth2 und Verschlüsselung mittels HTTPS gelöst werden.

Der Webservice stellt die Vermietungsplattform für einen fiktiven Bikesharingdienst dar.

## **2     INSTALLATION**

### **2.1   DATENBANK**

Für die Installation der Datenbank müssen die mitgelieferten Dumps `bikesharingservice.sql` und `bikesharingservice_oauth.sql` in eine MySQL Datenbank eingefügt werden. Anschließend muss die Datenbank gestartet werden.

### **2.2   WEBSERVICE UND API**

## 3 FUNKTIONEN WEBSERVICE

Name	Method	URL	Access
Alle verfügbare Fahrradstationen	GET	/stations	public
Spezielle Station	GET	/stations/stationID	public
Alle verfügbaren Fahrräder	GET	/bikes	public
Spezielles Fahrrad	GET	/bikes/bikesID	public
Alle Fahrradmodelle	GET	/models	public
Spezielles Fahrradmodell	GET	/models/modelID	public
Alle Buchungen	GET	/bookings	protected
Buchung erstellen	POST	/bookings	protected
Einzelne Buchung	GET	/bookings/bookingID	protected
Einzelne Buchung stornieren	DELETE	/bookings/bookingID	protected
Einzelne Buchung bearbeiten	PUT	/bookings/bookingID	protected
Accountinformationen	GET	/account	protected

### 3.1 GET METHODEN

#### 3.1.1 stations

Liefert eine Liste aller verfügbaren Stationen zurück an denen Fahrräder zum Verleih zur Verfügung stehen.

Method	URL	Access
GET	/stations	public

#### Parameter

Name	Required	Description
location	nein	Standort an dem nach Stationen gesucht werden soll, z.B. "Dresden" oder "Berlin"
model	nein	ID eines bestimmten Fahrradmodells nach dem gefiltert werden soll, z.B. 105 oder 185

Name	Required	Description
------	----------	-------------

## Request Example

GET /stations?location=Dresden&model=105

## Response

Liefert eine Liste von verfügbaren Stationen zurück, jede Station besteht aus den folgenden Parametern: \* stationId - Die einzigartige Kennung der Station \* name - Der Name der Station \* longitude - Längengrad \* latitude - Breitengrad \* bikes - Die Anzahl an zum Verleih zur Verfügung stehenden Fahrräder in der Station \* description - Die ausführliche Beschreibung der Station \* pictureUrl - Die URL eines Fotos von der Station

## Response Examples

```
{
  "stations" : [
    {
      "stationId" : 64,
      "name" : "Hauptbahnhof",
      "longitude" : -33.8670522,
      "latitude" : 151.1957362,
      "bikes" : 78,
      "description" : "Tolle Station am Hauptbahnhof, direkt vor dem Ausgang.",
      "pictureUrl" : "www.test.de/station64.jpg"
    },
    {
      "stationId" : 82,
      "name" : "Zoo",
      "longitude" : -33.8670522,
      "latitude" : 151.1957362,
      "bikes" : 108,
      "description" : "Eine Station direkt am Zoo.",
      "pictureUrl" : "www.test.de/station82.jpg"
    }
  ]
}
```

### 3.1.2 bikes

Liefert eine Liste aller verfügbaren Fahrräder die zum Verleih zur Verfügung stehen.

Method	URL	Access
GET	/bikes	public

## Parameter

Name	Required	Description
location	ja	Standort an dem nach verfügbaren Fahrrädern gesucht werden soll, z.B. "Dresden" oder "Berlin"
radius	nein	Radius in dem um die location gesucht werden soll in Meter (default: 2500), z.B. 5000
modelId	nein	Die eindeutige Kennung eines bestimmten Fahrradmodells nach dem gefiltert werden soll, z.B. 105
stationId	nein	Die eindeutige Kennung einer bestimmten Verleihstation nach der gefiltert werden soll, z.B. 4

## Request Example

GET /bikes?location=Dresden&radius=5000&modelId=105&stationId=4

## Response

Liefert eine Liste von verfügbaren Fahrrädern, jedes Fahrrad besteht aus den folgenden Parametern: \* bikeId - Die einzigartige Kennung des Fahrrads \* modelId - Die einzigartige Kennung des Fahrradmodells \* price - Der Preis für 15 Minuten in Cent. \* longitude - Längengrad \* latitude - Breitengrad \* stationId - Falls das Fahrrad in einer Verleihstation steht, ist dies die eindeutige Kennung der Station \* distance - Die Entfernung zum gewählten Standort

## Response Examples

```
{
  "bikes" : [
    {
      "bikeId" : 46,
      "modelId" : 105,
      "price" : 100,
      "longitude" : -33.8670522,
      "latitude" : 151.1957362,
      "stationId" : 4,
      "distance": 2800
    },
    {
      "bikeId" : 34,
      "modelId" : 105,
      "price" : 150,
      "longitude" : -33.8670522,
      "latitude" : 151.1957362,
      "stationId" : 4 ,
      "distance": 2800
    }
  ]
}
```

### 3.1.3 models

Liefert eine Liste aller verfügbaren Fahrradmodelle.



Method	URL	Access
GET	/models	public

## Parameter

Name	Required	Description
location	nein	Standort an dem nach verfügbaren Fahrrädern gesucht werden soll, z.B. "Dresden" oder "Berlin"
stationId	nein	Die eindeutige Kennung einer bestimmten Verleihstation nach der gefiltert werden soll, z.B. 4

## Request Example

GET /models?location=Dresden&stationId=4

## Response

Liefert eine Liste von verfügbaren Fahrradmodellen, jedes Modell besteht aus den folgenden Parametern: \* modelId - Die einzigartige Kennung des Fahrradmodells \* name - Der Name des Fahrradmodells \* description - Eine Beschreibung des Modells \* pictureUrl - Die URL eines Fotos, welches das Fahrradmodell zeigt \* bikes - Die Anzahl an zum Verleih zur Verfügung stehenden Fahrräder dieses Modells

## Response Examples

```
{
  "models" : [
    {
      "modelId" : 105,
      "name" : "Rennrad",
      "description" : "Bestens geeignet für das Fahren auf dem Asphalt.",
      "pictureUrl" : "www.test.de/model105",
      "bikes" : 78
    },
    {
      "modelId" : 68,
      "name" : "Kinderfahrrad",
      "description" : "Geeignet für Kinder zwischen 6 und 9 Jahren.",
      "pictureUrl" : "www.test.de/model68",
      "bikes" : 8
    }
  ]
}
```

### 3.1.4 account

Liefert die Accountinformationen des Nutzers.

Method	URL	Access
GET	/account	protected

### Request Example

GET /account

### Response

Liefert die Accountinformationen des Nutzers, ein Account besteht aus den folgenden Parametern: \* accountId - Die einzigartige Kennung des Nutzers \* email - Die E-Mail Adresse des Nutzers

### Response Examples

```
{
  "accountId" : 1696,
  "email" : "test@test.com"
}
```

## 3.1.5 bookings

Liefert eine Liste aller getätigten Buchungen.

Method	URL	Access
GET	/bookings	protected

### Request Example

GET /bookings

### Response

Liefert eine Liste aller getätigten Buchungen, jede Buchung besteht aus den folgenden Parametern: \* bookingId - Die einzigartige Kennung der Buchung \* bikeId - Die einzigartige Kennung des gebuchten Fahrrads \* date - Das Datum der Buchung

## Response Examples

```
{
  "bookings" : [
    {
      "bookingId" : 1682,
      "bikeId" : 105,
      "date" : "2013-12-01 18:44:36"
    },
    {
      "bookingId" : 1696,
      "bikeId" : 168,
      "date" : "2013-12-01 18:45:24"
    }
  ]
}
```

## 3.2 POST METHODEN

### 3.2.1 bookings

Erstellt eine einzelne Buchung.

Method	URL	Access
POST	/bookings	protected

### Request Example

```
POST /bookings
Params: {
  "bikeId": 168
}
```

### Parameter

Name	Required	Description
bikeld	ja	Die einzigartige Kennung des Fahrrads, z.B. 168

### Response

Liefert eine einzelne getätigte Buchung, die Buchung besteht aus den folgenden Parametern: - bookingId - Die einzigartige Kennung der Buchung - bikeld - Die einzigartige Kennung des gebuchten Fahrrads - date - Der Zeitpunkt an dem die Buchung getätigt wurde

## Response Examples

```
{
  "bookingId" : 1696,
  "bikeId" : 168,
  "date" : "2013-12-01 18:45:24"
}
```

## 3.3 DELETE METHODEN

### 3.3.1 bookings

Storniert eine einzelne Buchung.

Method	URL	Access
DELETE	/bookings/bookingId	protected

## Request Example

DELETE /bookings/1696

## 3.4 PUT METHODEN

### 3.4.1 bookings

Ändert eine einzelne Buchung.

WICHTIG: Content-Type: application/x-www-form-urlencoded

Method	URL	Access
PUT	/bookings/bookingId	protected

## Request Example

```
PUT /bookings/bookingId
Params: {
  "bikeId": 168
}
```

## Parameter

Name	Required	Description
bikeld	nein	Die einzigartige Kennung des Fahrrads, z.B. 168

## Response

Liefert die geänderte Buchung, die Buchung besteht aus den folgenden Parametern: - bookingId  
- Die einzigartige Kennung der Buchung - bikeld - Die einzigartige Kennung des gebuchten  
Fahrrads - date - Der Zeitpunkt an dem die Buchung getätigt wurde


## Response Examples

```
{  
  "bookingId" : 1696,  
  "bikeId" : 168,  
  "date" : "2013-12-01 18:45:24"  
}
```

## **4 WEBCLIENT**

Der Webclient ist so konzipiert, dass prinzipiell auch mobile Endgeräte unterstützt werden. Er bietet alle Funktionen die im vorhergehenden Kapitel beschrieben sind.

B i k e S h a r i n g



[STARTSEITE](#) [SUCHE](#) [STANDORTE](#) [MODELLE](#) [LOGIN](#)

### Suche

Adresse:

Postleitzahl

Stadt

Straße

Hausnummer

Umkreis in Metern:

Umkreis

Station:

wähle eine Station

Alle Angaben sind optional.

Suche Starten

[Kontakt](#) [Impressum](#)

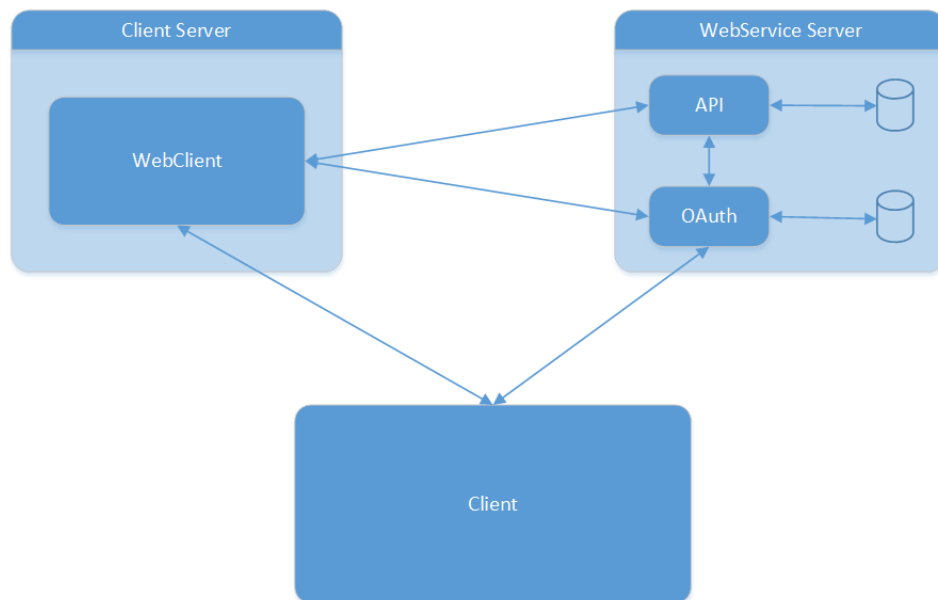
# **5 ZUSATZAUFGABE WS-SECURITY**

## **5.1 OAUTH2**

Das Architekturkonzept von OAuth2 wird durch folgendes Schema verdeutlicht.

## **5.2 IMPLEMENTIERUNG**





## 6 FAZIT

Die Implementierung des Webclient ist nach Vorlage einer durchdachten API gut machbar.

Das Slim-Framework war eine gute Wahl, da die Verwendung sehr einfach und fehlerfrei verlief. Die OAuth-Middleware hat leider nicht funktioniert, auch ist die Implementierung eines OAuth-Servers relativ kompliziert.

Der Aufwand ist, je nach gewählter Aufgabe, recht hoch, insbesondere wenn man, wie in diesem Fall, Programmiersprachen wählt, die den Teammitgliedern gar nicht, oder nur rudimentär geläufig sind. Wir hatten uns ursprünglich für PHP entschieden, da wir eine neue Programmiersprache kennenlernen wollten.

## **7 QUELLEN**