



Bike-Sharing Webservice mit Client-Applikation

BIKE-SHARING WEBSERVICE MIT CLIENT-APPLIKATION

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

CONTENTS

| | | |
|-------|---------------------------------------|----|
| 1 | Aufgabe | 3 |
| 2 | Installation | 4 |
| 2.1 | Datenbank | 4 |
| 2.2 | Webservice und OAuth-Server | 4 |
| 2.3 | Webclient | 4 |
| 3 | Webservice Endpoints | 5 |
| 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 | 10 |
| 3.3.1 | bookings | 10 |
| 3.4 | PUT Methoden | 11 |
| 3.4.1 | bookings | 11 |
| 4 | Webservice Implementierung | 12 |
| 5 | Webclient | 13 |

| | | |
|-----|---------------------------|----|
| 6 | Zusatzaufgabe WS-Security | 15 |
| 6.1 | OAuth2 | 15 |
| 7 | Fazit | 16 |

1 AUFGABE

Die Aufgabe bestand in einer Implementierung eines Webservices. Wir haben uns dabei für die Verwendung der Programmiersprache PHP entschieden. PHP hat den Vorteil, dass es auf nahezu jedem beliebigen Webserver ausführbar ist und relativ wenig Leistung benötigt. Dies wollen wir auch nach der Implementierung testen. Außerdem hatten wir uns für PHP entschieden, da wir gerne mal eine neue, uns eher unbekannte Programmiersprache kennenlernen wollten. Als Framework haben wir uns für Slim entschieden, ein Framework was sich vollständig auf die REST-Funktionalität beschränkt, trotzdem aber durch sogenannte Middlewares erweiterbar ist. Als Zusatzaufgabe wollten wir uns mit dem Thema Sicherheit befassen. Dafür sollte eine Authentifizierung mittels OAuth2 implementiert werden und die Verschlüsselung durch 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.

```
1 mysql -u root -p -h localhost
2 mysql> create database bikesharingservice;
3 mysql> create database bikesharingservice_oauth;
4 mysql> exit;
5 mysql -u username -p -h localhost bikesharingservice < bikesharingservice.sql
6 mysql -u username -p -h localhost bikesharingservice_oauth <
7     bikesharingservice_oauth.sql
```

Listing 2.1: Datenbankimport

Anschließend muss die Datenbank gestartet werden.

Alternativ können die Datenbanken auch über phpMyAdmin importiert werden.

2.2 WEBSERVICE UND OAUTH-SERVER

Vorraussetzung für den Webservice und den OAuth-Server ist ein Webserver auf dem PHP und PDO installiert sind. In den beiden Dateien `config.php` und `config_oauth.php` sind die Zugangsdaten für die Datenbank hinterlegt, diese müssen entsprechend angepasst werden. Anschließend kann der Order api einfach auf den Server kompiert werden.

2.3 WEBCLIENT

Für den Webclient ist ebenfalls ein Server mit installiertem PHP nötig. Zusätzlich muss in der `script.js` die Variable `baseURL` durch die URL des Webservice ersetzt werden. In der `head.php` muss dies ebenfalls über die Variable `$api_url` geschehen. Durch die Variable `$oauth_url` wird zusätzlich der Link zum Authentifizierungsserver gesetzt.

3 WEBSERVICE ENDPOINTS

| 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 |

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. 64 |

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

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 |
|--------|----------|---|
| bikeId | 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 - bikeId - 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 |
|--------|----------|---|
| bikeId | 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 - bikeId - 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 WEBSERVICE IMPLEMENTIERUNG

Im Folgenden soll grundlegend erläutert werden wie sich ein Webservice mit dem Slim-Framwork implementieren lässt. Jeder Request wird dabei wie folgt entgegengenommen:

```
1 $app->get('/bikes/:bikeId', function ($id) use ($app) {  
2     $query = "SELECT id, model, price, longitude, latitude, station  
3         FROM bikes WHERE id = $id";  
4     echo getObjectFromDb($query, $app);  
5 });
```

Listing 4.1: GET-Request mit Slim

In dem Beispiel wird durch das Wort `get` festgelegt, dass GET-Requests entgegengenommen werden soll, durch das Ersetzen mit `post`, `put` oder `delete`, würden entsprechend die Arten von Requests entgegengenommen werden. Anschließend wird die URL definiert, wobei der Doppelpunkt angibt, dass es sich um eine Variable handelt. In der Funktion wird die SQL-Query erstellt und die von uns erstellte Funktion `getObjectFromDb()` aufgerufen. Diese führt die Query mittels PDO aus und liefert das Ergebnis als JSON zurück. Die Umwandlung des Ergebnisses in JSON erfolgt durch die PHP-Funktion `json_encode()`.

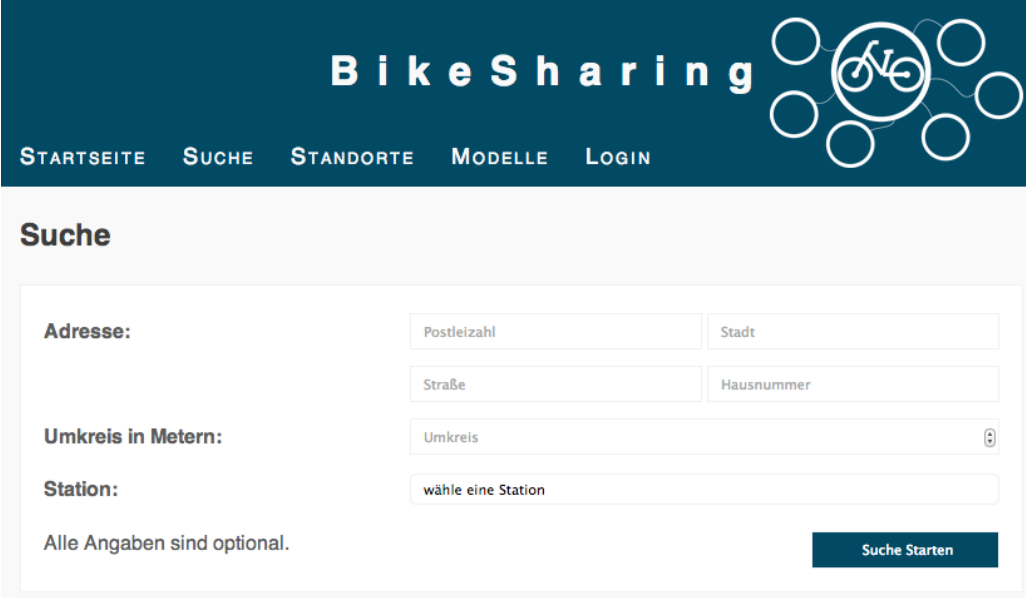
Auf die Art und Weise wurden alle Endpoints implementiert, zusätzliche Parameter lassen sich in der Funktion wie folgt abrufen:

```
1 $app->request()->get('stationId');
```

Listing 4.2: Paramater mit Slim

5 WEBCLIENT

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



The screenshot shows the 'Bike Sharing' web client interface. At the top is a dark blue header with the text 'Bike Sharing' in white, followed by a logo of a bicycle inside a circle surrounded by smaller circles. Below the header is a navigation bar with links: 'STARTSEITE', 'SUCHE', 'STANDORTE', 'MODELLE', and 'LOGIN'. The main content area is titled 'Suche' (Search). It contains a search form with the following fields: 'Adresse:' with sub-fields for 'Postleitzahl' (postal code), 'Stadt' (city), 'Straße' (street), and 'Hausnummer' (house number); 'Umkreis in Metern:' (radius in meters) with a 'Umkreis' (radius) field and a dropdown arrow; 'Station:' with a 'wähle eine Station' (select a station) dropdown. Below the form, it says 'Alle Angaben sind optional.' (All data is optional). A dark blue button labeled 'Suche Starten' (Start Search) is at the bottom right of the form.

Figure 5.1: Suche nach Fahrrädern

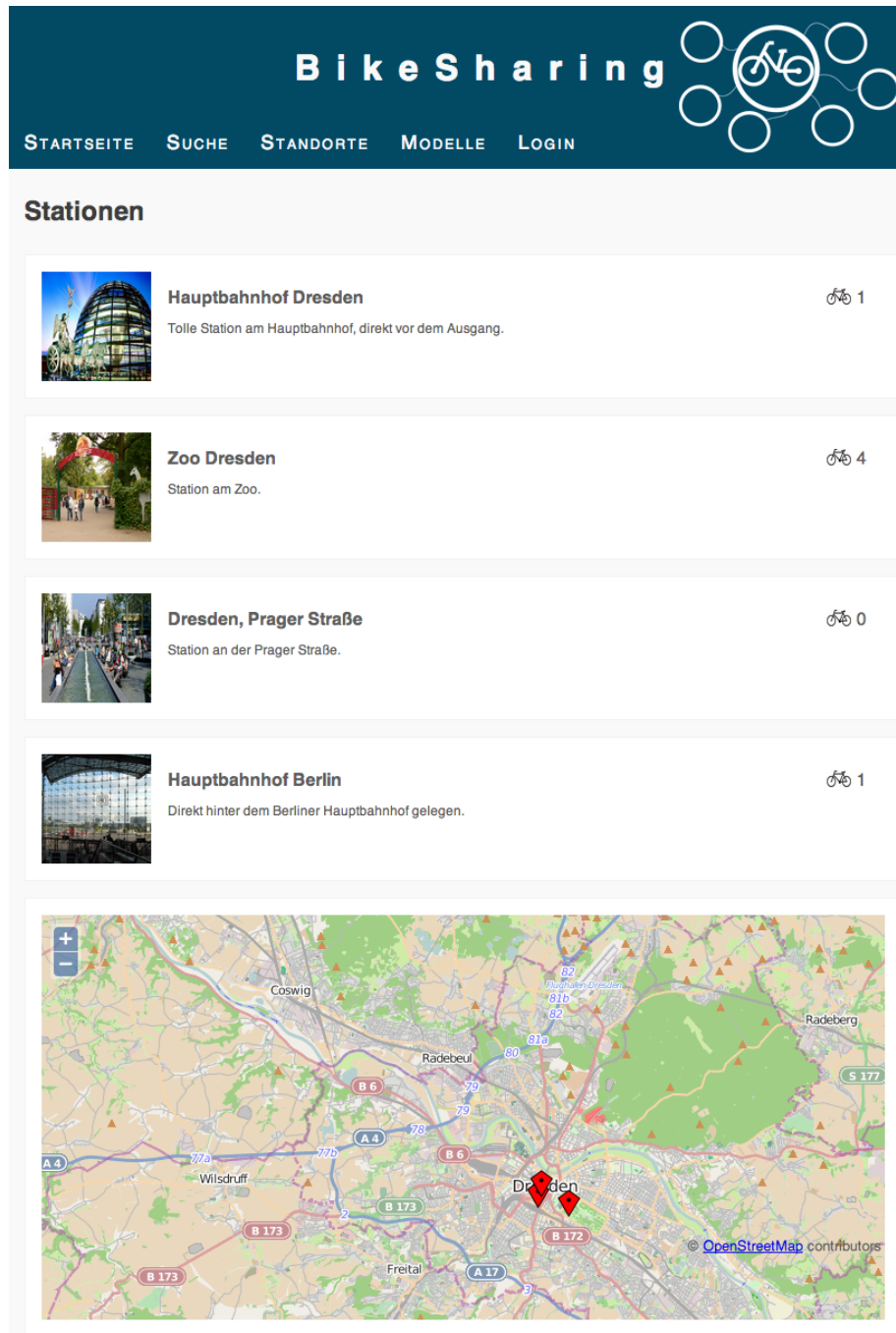
Suche: Es kann nach Fahrrädern an bestimmten Adressen oder Stationen gesucht werden. Nach dem Absenden der Suche werden alle gefundenen Fahrräder aufgelistet und jeweils die Entfernung zum angegebenen Standort angezeigt.

Stationen: Alle Stationen werden zum einen in einer Liste, zum anderen auf einer Karte angezeigt. Beim Auswählen einer Station wird diese noch einmal im Detail angezeigt, inklusive dem Standort auf einer Karte und allen verfügbaren Fahrrädern.

Modelle: Alle Modelle werden in einer Liste dargestellt. Beim Auswählen eines Modells wird dieses noch einmal im Detail angezeigt, inklusive allen verfügbaren Fahrrädern.

Profil: Im Profil werden zum einen die Daten des Nutzers angezeigt, zum anderen alle getätigten Buchungen. Buchungen können hier auch storniert werden.

Fahrrad: Für jedes Fahrrad existiert eine Detailansicht, wo das Modell und der Standort auf einer Karte angezeigt werden. Zusätzlich lässt sich hier das Fahrrad buchen.



6 ZUSATZAUFGABE SICHERHEIT

Wie in Abbildung 6.1 dargestellt, besteht unser Webservice aus zwei Teilen. Neben der eigentlichen API, gibt es zusätzlich noch eine Komponente die für die Authentifizierung zuständig ist, den OAuth-Server. Dieser hat auch seine eigene Datenbank. Nachdem der Nutzer auf der Clientanwendung, in unserem Fall dem Webclient, auf eine Seite zugreifen möchte, die eine Authentifizierung erfordert, erfolgt eine Weiterleitung zum OAuth-Server. Dort muss der Nutzer sich dann mit seinen Zugangsdaten anmelden und bestätigen, dass die Clientanwendung auf die privaten Daten zugreifen darf. Daraufhin wird der Nutzer wieder zum Webclient weitergeleitet und dabei die Authentification-Code übergeben, der vom OAuth-Server generiert wurde. Diesen Code verwendet die Clientanwendung zusammen mit ihren eigenen Zugangsdaten um beim OAuth-Server einen Access-Token anzufordern. Mit diesem Access-Token kann die Clientanwendung nun auf die privaten Daten des Nutzers zugreifen.

Um zu verhindern das jemand unautorisiertes den Access-Token abgreift, wird für die komplette Kommunikation HTTPS verwendet. Zusätzlich erfolgen alle Abfragen an den Webservice direkt vom Server des Webclient aus, der Access-Token wird dort in der Session gespeichert. Dadurch ist die unsichere clientseitige Speicherung mit JavaScript nicht nötig.

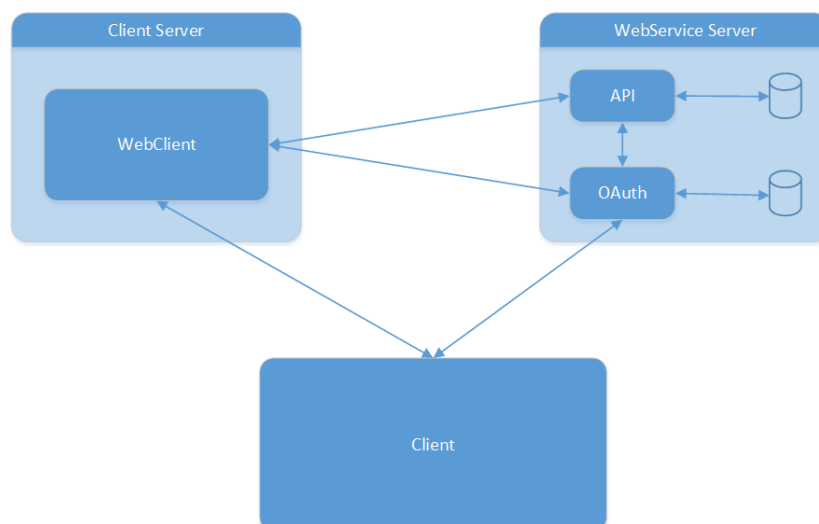


Figure 6.1: Kommunikation zwischen, Webservice, Client und Nutzer

7 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 Implementierung eines OAuth-Servers ist recht zeitaufwendig und umständlich, vor allem wenn man sich damit bisher noch nicht befasst hat. Leider hat auch die OAuth-Middleware für das Slim-Framework nicht funktioniert, sodass eigene Lösung gefunden werden musste, um die Authentifizierung auf dem Webservice zu kontrollieren.

Der Aufwand ist, ja nach gewählter Aufgabe, recht hoch, insbesondere wenn man, wie in diesem Fall, Programmiersprachen wählt, die den Teammitgliedern gar nicht, oder nur rudimentär geläufig sind.

Wie am Anfang geplant, konnten wir unsere Anwendung nach der Entwicklung problemlos auf einem fast kostenlosem Webserver testen und sie ließ sich dort direkt ohne Probleme ausführen, es war keine besondere Einrichtung erforderlich.