Bachelorarbeit – Softwareentwicklung und Wirtschaft

A web-based tracking system for the public transport

Christoph Maurer

TU Graz, Sommersemester 2014

1. Einleitung und Motivation

Öffentliche Verkehrsmittel sollten grundsätzlich pünktlich sein und nach Fahrplan fahren. Mit der Ausnahme von vollautomatisierten U-Bahnsystemen wird es aber nie exakte Pünktlichkeit geben und Verspätungen kommen daher leider regelmäßig vor. Besonders im ländlichen Raum, wo sich der öffentliche Verkehr auf Busse beschränkt, sind durchschnittliche Verspätungen von fünf Minuten normal. Erschwerend kommt hinzu, dass diese Busse oft nur selten verkehren und daher ein Versäumen für die Betroffenen besonders schwer zu tragen kommt.

Ein Informationssystem schafft natürlich nicht direkt mehr Busse und es wird auch nicht sofort bessere Pünktlichkeit geben. Mit dieser Arbeit möchte ich aber ein Informationssystem schaffen, welches den Standort von Bussen in Echtzeit erfasst und Fahrplanabweichungen rechtzeitig erkannt werden können. Dieses System wird in weiterer Folge ***Oeffitrack*** genannt werden.

2. Problemstellungen und Lösungsansätze

Aus technischer Sicht stellt sich die Frage wie die Standorte der Busse erfasst und dem Passagier, welcher den Bus rechtzeitig erreichen möchte präsentiert werden können.

Eine ausreichend genaue Standorterfassung ist in der heutigen Zeit dank GPS leicht geworden. Mobiles Internet hat sich in den letzten Jahren stark verbreitet (auch im ländlichen Raum) und ein Smartphone haben viele Menschen in ihrer Hosentasche.

Oeffitrack wurde als Webapplikation entwickelt. Das bedeutet hier, dass sowohl die Standorterfassung der Busse, als auch die Visualisierung für den Benutzer der öffentlichen Verkehrsmittel von der selben Applikation durchgeführt wird. Dank der Geolocation-API von HTML5 ist das problemlos möglich. Das verschafft gleich mehrere Vorteile. Einerseits vereinfacht diese Tatsache die Entwicklung, da immer nur in der Codebasis der Webapplikation gearbeitet werden muss und für die Geräte in den Fahrzeug erhält man so den Vorteil der Plattformunabhängigkeit, denn man braucht ja nur ein Gerät mit einem

Browser der Geolocation unterstützt. Die Visualisierung der Fahrplandaten erfolgt ebenfalls im Browser. Für viele Benutzer wäre es sicher angenehmer, wenn sie eine speziell an ihr Smartphone angepasste App hätten. Oeffitrack wurde zwar als Webapplikation entwickelt, es wurde jedoch bereits darauf Wert gelegt, dass sämtliche Daten auch maschinenlesbar von der Webapplikation im JSON-Format abgefragt werden können. Wenn Oefftirack also in Zukunft großflächig eingesetzt werden sollte wird es daher leicht werden spezielle Apps dafür zu entwickeln. Ein tolles Szenario wäre wohl eine Smartwatchanwendung welche zu blinken und vibrieren beginnt um so zu signalisieren, dass man zu laufen beginnen sollte falls man den Bus doch noch erreichen möchte.

Oeffitrack wurde als klassische **PHP-MySQL** Anwendung entwickelt. Serverseitig wurde auf das bekannte Webframework **CodeIgniter** gesetzt. Damit werden wiederkehrende Aufgaben wie Datenbankzugriff, Sessionmanagement und URL-Mapping stark vereinfacht. Weiters wird man zur Verwendung der Model-View-Controller Architektur gedrängt und so soll automatisch besserer Code entstehen als würde man klassisch mit PHP programmieren und die Trennung zur Darstellungsschicht nicht ganz so streng nehmen.

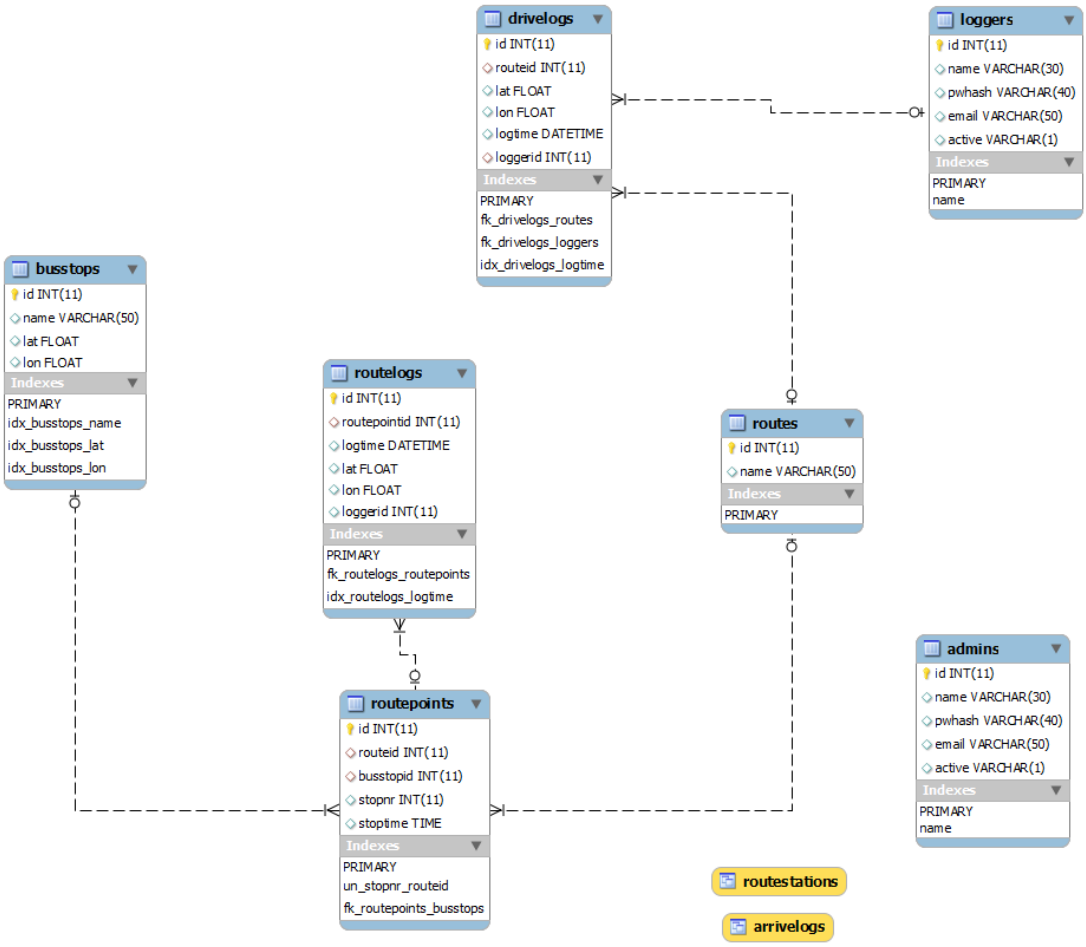
Die Fahrplandaten werden im Browser in Tabellenform und in Karten dargestellt. Wichtig war hier die automatische Aktualisierung der Daten. Für die Karten wurde auf die Google Maps API gesetzt und zur einfacheren und portableren Verwendung von Javascript auf das weitverbreitete jQuery.

Die Projektziele für diese Arbeit waren zusammengefasst:

- Möglichkeit zur Anzeige und Verwaltung der Fahrpläne
- Anzeige der Echtzeitinformationen auf Karten und in Tabellen
- Erfassung der Echtzeitdaten für autorisierte Benutzer (Busfahrer)
- Einfacher Zugriff auf die Daten für andere Applikationen

3. Datenbankschema

Dieser Abschnitt beschreibt das entwickelte Datenbankschema, also die Tabellen und Views die für Oeffitrack benötigt werden.



Obige Abbildung zeigt das Datenbankschema exportiert von MySQL-Workbench als EER-Model. Daraus lassen sich bereits die wichtigsten Abhängigkeiten ersehen. Trotzdem ist die Bedeutung aller Tabellen nicht sofort klar, daher werden sie nun erklärt.

Tabellen

Busstops:

Diese Tabelle repräsentiert die Haltestellen welche in Oeffitrack neben dem Namen, eine genaue Position in Breitengrad (Lat/Latitude) und Längengrad (Lon/Longitude) benötigen.

Routes:

In Routes werden einzelne Linien verwaltet. Als Linie versteht man hier eine Strecke, die zu einer bestimmten Uhrzeit einmal täglich gefahren wird.

Routepoints:

Unter den Routepoints versteht man einzelne Stationen einer Strecke. Eine Station ist eine Haltestelle, die zu einer gewissen Uhrzeit (stoptime) erreicht werden soll.

Routelogs:

In der Tabelle Routelogs sind die Ist-Daten der erreichten Routepoints gespeichert.

Drivelogs:

Hier werden geloggte Koordinaten einer Strecke gespeichert. Im Gegensatz zur Tabelle Routelogs sind hier alle Koordinaten einer Strecke gespeichert. Die Trennung in diese beiden Tabellen Routelogs und Drivelogs war eine Designentscheidung. Das hat einerseits Performancegründe, andererseits kann man so viel leichter abfragen wann eine konkrete Haltestation erreicht wurde. Die Tabelle Drivelogs möchte man wahrscheinlich früher leeren bzw. archivieren, da die Datenmenge hier schnell anwachsen wird, da ja periodisch die Koordinaten geloggt werden. Die Werte in Routelogs sind länger interessant um zum Beispiel einen erstellten Fahrplan an die IST-Daten anzupassen oder andere statistische Auswertungen durchführen möchte.

Loggers:

Hier werden die Benutzer vermerkt, welche die Logdaten (Drivelogs) erzeugen. Die Passwörter werden als sha1-Hash gespeichert.

Admins:

Spezielle Benutzer zur Verwaltung des Systems. Auch hier sind die Passwörter als sha1-Hash gespeichert.

Views

Um die Fahrpläne und Soll-Ist-Daten leicht abfragen zu können wurden neben den Tabellen zwei Views erstellt.

Routestations:

Mit dieser View erhält man den Fahrplan, also alle Haltestellen und Abfahrtszeiten einer Strecke.

Arrivelogs:

Aus dieser View kann man die Soll-Ist-Daten leicht abfragen ohne jedes mal komplizierte Joins schreiben zu müssen.

4. Einführung in CodeIgniter und Anzeigen von Routen

In diesem Abschnitt wird am Beispiel der Linienübersicht in die Verwendung von CodeIgniter eingeführt.

```php
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Routes extends CI_Controller
{
  function index($view = 'html', $offest = -1, $limit = -1)
  {
    $this->db->select('id,name');
    $this->db->order_by('name', 'asc');
    if ($offest != -1 && $limt != -1)
    {
      $this->db->limit($offest, $limit);
    }
    $query = $this->db->get('routes');
    $rows = $query->result_array();
    if ($view == 'json')
    {
      $this->output->set_content_type('application/json');
      $this->output->set_output(json_encode($rows));
    }
    elseif ($view == 'html')
    {
      $this->load->view('stdheader');
      $this->load->view('routes', array('rows' => $rows));
      $this->load->view('stdfooter');
    }
  }
}
```

Obiges Beispiel zeigt den Controller der für das Anzeigen der Linien eingesetzt wird. Die Namensgebung der Klasse (hier Routes) und Methoden (hier nur index) bestimmen die URL unter der die Seiten aufrufbar sind. Die Parameter von `index()` sind dank dem Apache-Modul **mod_rewrite** ebenso ein Teil der URL. Da der Parameter `$view` per default auf „html“ gesetzt wird kann die HTML-Ausgabe unter der URL http://example.com/routes/ angezeigt werden. Wie bereits erwähnt wurde darauf geachtet, dass sämtliche Daten auch als JSON abgefragt werden können. Dies geht hier mittels http://example.com/routes/index/json. Auffallend ist hier, dass im Gegensatz zu vorher auch index angegeben werden muss. Die Methode `index()` wird per default aufgerufen, muss aber angeben werden, wenn dahinter noch weitere Parameter (hier json) angegeben werden. Mit den optionalen Parametern `$offset` und `$limit` kann man die Ergebnismenge einschränken. Wie bereits erwähnt erleichtert CodeIgniter die Verwendung der Model-View-Controller Architektur. Bei diesem einfachen Beispiel sieht man aber, dass auf die Verwendung eines Models verzichtet wurde. Dies wäre hier ein nicht notwendiger Mehraufwand, da dieser Controller nur die Linien ohne weiterer Logik lädt. Aufgrund der in CodeIgniter integrierten Datenbankunterstützung können die Linien mittels `$this->db->get('routes')` selektiert werden ohne eigenes SQL schreiben zu müssen.

Views befinden sich bei einer CodeIgniter-Applikation unter `application/views`. Eine View selbst ist in CodeIgniter eine einfache PHP-Datei. Eine eigene Templatesprache ist in der Grundinstallation nicht integriert und wird von Oeffitrack auch nicht verwendet.

```
<h1>Routes</h1>

<div id="routesdiv">
<ul>
<?php foreach($rows as $row): ?>
<li><a href="/route/index/<?php echo $row['id'];?>"><?php echo $row['name'].' / '.$row['id'];?></a></li>
<?php endforeach; ?>
</ul>
</div>
```

Obiger Code zeigt die View zur Anzeige der Linien. Die View selbst wird vom Controller mittels `$this->load->view('routes', array('rows' => $rows))` geladen. Man sieht hier die Datenübergabe an die View als Key-Value-Array.

Man sieht hier die HTML-Ausgabe der Routen. Hier gibt es nur eine Strecke „Nestelbach-Großwilfersdorf“ mit der Streckennummer (Id) 2. Dieser Link führt zur Detailansicht dieser Route.
Für die JSON-Ausgabe wird keine eigene View-Datei benötigt. Wie dem Controller zu entnehmen ist wird einfach der Content-Type auf „application/json“ festgelegt und das Rows-Array als JSON ausgegeben. Man erhält hier folgenden Output.

```
[{"id":"2","name":"Nestelbach-Gro\u00dfwilfersdorf"}]
```

5. Erfassen der Echtzeitdaten

Das Erfassen der Echtzeitdaten ist das Kernstück von Oeffitrack. Eine Designentscheidung war es sämtliche Aufgaben von Oeffitrack, insbesondere auch das

Logging der Positionen der Busse als Webapplikation zu realisieren. In diesem Abschnitt wird die konkrete Umsetztung davon beschrieben.

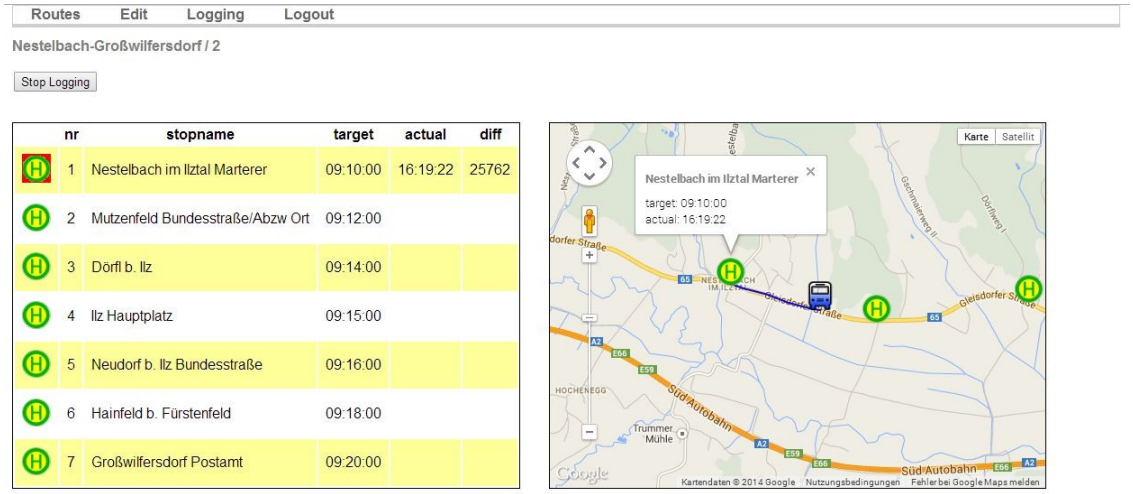
Durch die Geolocation-API kann man über Javascript die aktuellen Koordinaten im Browser abfragen. Diese Koordinaten werden mittels AJAX periodisch an den Webserver übertragen und damit hat man schon die meisten technischen Probleme überwunden. Alternativ wird serverseitig eine XML-RPC-Schnittstelle angeboten um leichter Drittapplikationen für Oeffitrack schreiben zu können.

Die aktuelle Position wird mittels Javascript in Oeffitrack folgendermaßen abgefragt:

```
navigator.geolocation.getCurrentPosition(success, error, options);
```

success ist eine Callback-funktion, welche falls die Koordinaten erfolgreich bestimmt werden konnten, die Position mittels AJAX an den Webserver schickt. Im Fehlerfall (falls die aktuelle Position gerade nicht bestimmt werden konnte) wird die Funktion error aufgerufen und eine Fehlermeldung mit dem entsprechenden Errorcode in der Javascript-Console ausgegeben. options ist ein Key-Value-Array zur Konfiguration der Geolocation API. Hier wird im Wesentlichen bestimmt, dass keine zu alten Koordinaten verwendet werden. Der Benutzer könnte sonst fälschlicherweise glauben, dass sich der Bus noch weiter entfernt befindet als er es wirklich ist.

Folgende Abbildung zeigt das Logging der Positionen im Browserfenster.



In dieser Abbildung sieht man auf der linken Seite die Haltestationen mit Plan- (target) und

Ist-Zeiten (actual). Wurde eine Haltestelle zu spät (mindestens 3 Minuten Verspätung) erreicht wird das Haltestellensymbol in der Tabelle rot eingefärbt. Im Bild wurde erst die erste Haltestelle erreicht mit einer (hoffentlich) unrealistisch hohen Verspätung. Auf der rechten Seite sieht man die Darstellung der Echtzeitdaten in den Google Maps. Im Gegensatz zur Tabelle sieht man hier noch genauer wo sich der Bus gerade befindet, da in der Tabelle ja nur die Ankunftszeiten bzw. Sollzeiten eingetragen sind. Damit lässt sich für den Benutzer zum Beispiel erkennen ob der Bus gerade im Stau steht oder ob etwa eine Umleitung benutzt werden musste. Durch einen Klick auf die Haltestellensymbole auf der Karte sieht man ebenfalls die Soll-Ist-Daten. Jede geloggte Position ist auf der Karte als Blaue Linie zu sehen.
Der Code der auf Seite des Clients welcher für das Loggen verantwortlich ist findet sich im Ordner views in der Datei logtool.php.
Hier sei kurz gezeigt wie einfach das Logging in Javascript auch dank jQuery funktioniert.

```
218 function logPosition(routeid, current_lat, current_lon)
219 {
220   var rpid = -1;
221   $.each(routestations, function(i, rs) {
222     if (rs.logged == false &&
223       getDistanceFromLatLonInMeter(rs.lat, rs.lon, current_lat, current_lon) <= 60.0)
224     {
225       rpid = rs.routepointid;
226       rs.logged = true;
227       return false;
228     }
229   });
230
231   $.post("/logging/log/",
232       {
233         routeid: routeid,
234         lat: current_lat,
235         lon: current_lon,
236         routepointid: rpid
237       },
238     function(data, textStatus) {
239     //data contains the JSON object
240     //textStatus contains the status: success, error, etc
241     }, "json");
242
```

Diese Funktion `logPosition()` wird im Sucess-Callback von `getCurrentPosition()` und dieser im Intervall von 10 Sekunden falls keine Fehler auftreten aufgerufen. Dieser Zeitabstand sollte für eine ausreichend hohe Genauigkeit sorgen. In `logPosition()` ist zu sehen, dass eine Haltestelle als erreicht markiert wird wenn sich der Bus im Umkreis von 60 Metern der Haltestelle befindet. Der Client überträgt dann die Routepointid der erreichten Haltestelle an den Server. Ansonsten wird für die Routepointid der Wert -1 übertragen. Der Server speichert die erfassten Koordinaten dann nur in die Tabelle Drivelogs und nicht in die Routelogs. Die Erkennung, dass eine Haltestelle erreicht wurde

wird deswegen vom Client durchgeführt, weil es ihm einerseits sowieso bekannt ist und weiters der Server so entlastet werden kann, da dieser keinen State darüber halten muss und so einerseits einfacher zu programmieren ist und dadurch vor allem bessere Performance erreicht werden kann.
Dem Post-Request (Zeile 231) in obigen Code ist der Controller zu entnehmen den die Positionsdaten geschickt werden. Aus dem vorigen Abschnitt ist bekannt, dass es sich dabei um die Controllerklasse `Logging` und die Methode `log` handeln muss. Folgender Code zeigt diese Methode.

```
32  function log()
33  {
34    if (!$this->session->userdata('logged_in')) {
35      die('not logged in');
36    }
37    $this->load->model('positionlogger');
38
39    $logger =  $this->session->userdata('logger');
40    $loggerid = $logger['id'];
41    $routeid = $this->input->post('routeid', TRUE);
42    $lat = $this->input->post('lat', TRUE);
43    $lon = $this->input->post('lon', TRUE);
44    $routepointid = $this->input->post('routepointid', TRUE);
45
46    $rv = $this->positionlogger->logPosition($loggerid, $routeid, $lat, $lon, $routepointid);
47    $this->output->set_content_type('application/json');
48    $this->output->set_output(json_encode(array('status' => 'OK')));
49  }
50
```

Zu sehen ist hier einerseits die Prüfung ob der User eingeloggt ist, da ja nur autorisierte User berechtigt sind Postionen zu senden. Und weiters sieht man, dass das eigentliche Logging an das Model `Positionlogger` weiter delegiert wird. Als Output bekommt man ein in JSON verpacktes „OK“.

```php
class PositionLogger extends CI_Model
{
  function logPosition($loggerid,$routeid,$lat,$lon,$routepointid=-1)
  {
    $rv = true;
    log_message('debug','trying logPosition()');
    $logtime = date('Y-m-d H:i:s');
    $data = array(
          'loggerid' => $loggerid,
          'routeid' => $routeid,
          'lat' => $lat,
          'lon' => $lon,
          'logtime' => $logtime
        );

    $rv = $this->db->insert('drivelogs', $data);

    if  (!$rv) {
      $logdata = var_export($data,true);
      log_message('error','Insert into drivelogs failed: '.$logdata."\n");
    }

    if ($routepointid !=-1 && $rv) {
      $sdate = date('Y-m-d').' 00:00';
      $edate = date('Y-m-d H:i:s');//' 23:59:59';
      $this->db->where(array('routeid' => $routeid, 'routepointid' => $routepointid, 'logtime >=' => $sdate, 'logtime <=' => $edate));
      $count = $this->db->count_all_results('arrivelogs');
      if  ($count==0) {
        $data = array(
          'routepointid'=>$routepointid,
          'logtime'=>$logtime,
          'lat'=>$lat,
          'lon'=>$lon,
          'loggerid'=>$loggerid,
        );
        $rv = $this->db->insert('routelogs', $data);
        if (!$rv) {
          $logdata = var_export($data, true);
          log_message('error','Insert into routelogs failed: '.$logdata."\n");
        }
      } else {
        log_message('warn', 'Routelog for routepointid '.$routepointid." already exists\n");
      }
```

Hier in diesem Model sieht man nun das Einfügen in die Tabelle Drivelogs und falls die Routepointid ungleich -1 ist, das zusätzliche Einfügen in die Tabelle Routelogs.

Der Vollständigkeit halber sind hier die URLs zur Abfrage der Drivelogs und Arrivelogs (Soll-Ist-Daten) im JSON-Format angegeben:

- http://localhost/drivelogs/index/<routeid>


```
[{"lat":"47.0891","lon":"15.8954","logtime":"2014-06-09 16:20:42"},
{"lat":"47.0891","lon":"15.8948","logtime":"2014-06-09 16:20:32"},
{"lat":"47.0895","lon":"15.8917","logtime":"2014-06-09 16:20:22"},
{"lat":"47.0898","lon":"15.8902","logtime":"2014-06-09 16:20:12"},
{"lat":"47.0899","lon":"15.8895","logtime":"2014-06-09 16:20:02"},
{"lat":"47.0903","lon":"15.8872","logtime":"2014-06-09 16:19:52"},
{"lat":"47.0908","lon":"15.8852","logtime":"2014-06-09 16:19:42"},
{"lat":"47.0912","lon":"15.8829","logtime":"2014-06-09 16:19:32"},
{"lat":"47.0916","lon":"15.8813","logtime":"2014-06-09 16:19:22"}]
```


- http://localhost/arrivelogs/index/<routeid>


```
[{"stopnr":"2","name":"Mutzenfeld Bundesstra\u00dfe\/Abzw
Ort","lat":"47.089","lon":"15.8956","stoptime":"09:12:00","logtime":"2014-
06-09 16:20:42"},{"stopnr":"1","name":"Nestelbach im Ilztal
Marterer","lat":"47.0914","lon":"15.8817","stoptime":"09:10:00","logtime":
```

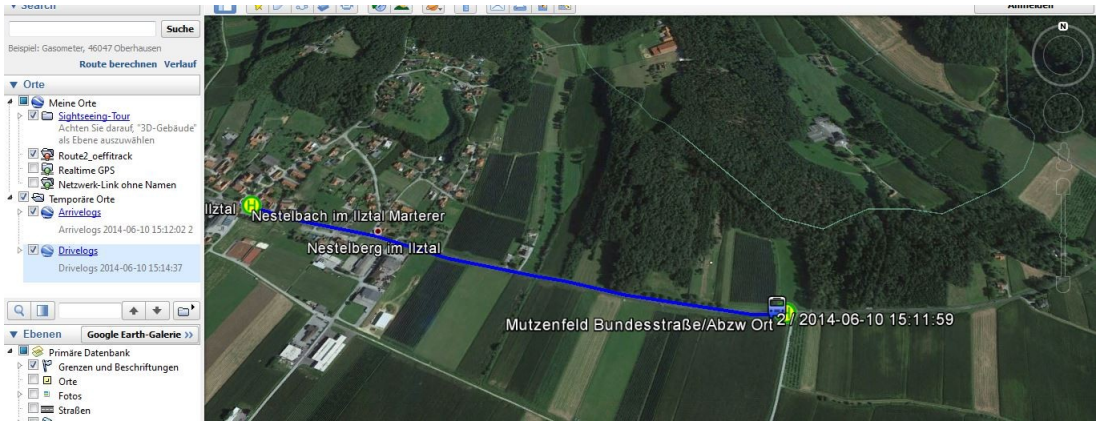
```
"2014-06-09 16:19:22"}]
```

localhost ist hier natürlich durch die URL der Oeffitrack-Installation zu ersetzen.
In views/logtool.php sieht man auch die Abfrage der Daten dieser URLs in Javascript.

Weiters sei erwähnt, dass die Positionen auch über XML-RPC übermittelt werden können. Hierfür kann über die URL http://localhost/logservice die Methode `LogService.log(loggerid, pw, routeid, lat, lon, routepointid)` aufgerufen werden.

6. Anzeigen der Soll-ist-Daten

Im Browser sieht die Anzeige der Soll-Ist-Daten für den Benutzer gleich aus wie für den eingeloggten Benutzer (logger) der die Daten zum Server schickt. Zusätzlich können die Drivelogs (periodisch empfangene Positionen der Busse) und Arrivelogs (Soll-Ist-Daten) über KML exportiert werden und so von einem Geoinformationssystem wie Google Earth angezeigt werden. Folgendermaßen können die Daten als KML empfangen werden:
http://localhost/arrivelogs/index/2/kml
bzw.
http://localhost/drivelogs/index/2/kml
Nachfolgende Abbildung zeigt diese KML-Daten in Google Earth.



In der Webapplikation hätten diese generierten KML-Daten ebenfalls von der Google Maps API direkt verwendet werden können. Die Anzeige dieser Daten wurde aber trotzdem manuell programmiert. Zum einen ist man so flexibler und ist nicht auf die Visualisierung des Defaultverhaltens von KML in den Google Maps abhängig. Zum anderen läuft die Google Maps API auf den Servern von Google und diese können logischerweise nicht ohne Weiteres auf den eigenen Rechner zugreifen was ein Problem ist wenn man auf localhost entwickelt.

7. Testen der Positionserfassung

Da man nicht immer echte Positionsdaten zu Verfügung hat wurde eine Möglichkeit geschaffen diese zu simulieren. Dafür wurde auf GeoMock (https://github.com/janmonschke/GeoMock) gesetzt. GeoMock ist eine einfache Javascript-Datei mit welcher man die Positionsdaten für `navigator.getCurrentPosition()` vorgeben kann. Um die vorgegebenen Positionsdaten zu aktivieren kann dem im vorigen Abschnitt beschriebenen „logtool“ ein Mockparameter übergeben werden.

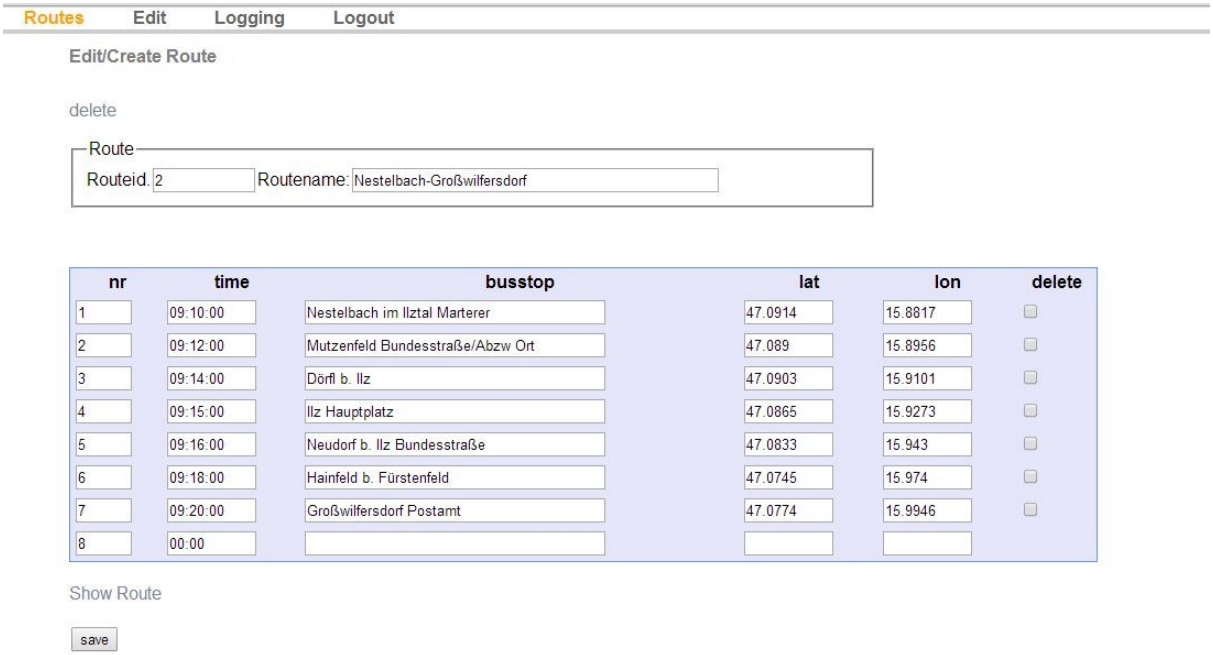
http://localhost/logging/logtool/2/1

Der letzte Parameter ist hier der Mockparameter, der auf 1 gesetzt ist. Damit werden für die Strecke 2, die vorgegebenen Geodaten verwendet.

Es lassen sich auch Verzögerungen und fehlerhafte Positionsabfragen simulieren.

8. Bearbeiten von Routen

Ein Entwicklungsziel war auch die Möglichkeit zum Verwalten von Linien zu schaffen. Folgende Abbildung zeigt das Interface welches für diesen Zweck erschaffen wurde.



Das Interface ermöglicht alle notwendigen Operationen (Create/Update/Delete) sowohl von der ganzen Strecke also auch von einzelnen Haltestationen.
Zugegeben ist dieses Formular nicht sehr komfortabel wenn man ein reales System mit tausenden Strecken aufsetzen bzw. Fahrplandaten aktualisieren möchte. Die Logik dahinter ist aber so gehalten, dass man sehr leicht zum Beispiel einen CSV-Import dazu ergänzen könnte.

9.Zusammenfassung und Ausblick

Im Zuge dieser Arbeit ist ein System mit dem Namen „Oeffitrack“ entstanden welches als reine Webapplikation realisiert wurde. Es wurde darauf geachtet, dass Echtzeitdaten automatisch im Browserfenster aktualisiert werden ohne manuellen Reload der Seite durch den Benutzer, da aktuelle Daten für eine solche Anwendung von besonderer Wichtigkeit sind.

Natürlich ist eine Webapplikation nicht für alle Situation die beste Wahl. Zwar sind heute bereits der Zugriff auf Geodaten durch den Browser möglich, aus Komfortgründen würde es sich aber für Endnutzer lohnen wenn für Oeffitrack spezielle Apps für Smartphones,

Smartwatches etc. entwickelt werden. Wenn man unter Stress steht und gerade zum Bus eilt hat man logischerweise meist nicht die Zeit das Browserfenster zu öffnen und nach dem Bus zu suchen. Deswegen wurde darauf geachtet, dass sämtliche Daten die von Oeffitrack erfasst und dargestellt werden auch maschinenlesbar im JSON-Format bzw. als KML abgefragt werden können. Falls Oeffitrack also einmal erfolgreich eingesetzt werden sollte kann man davon ausgehen, dass weitere Apps für das System entwickelt werden.

Eine Designentscheidung war es das Grundsystem möglichst schlank zu halten und das Datenbankschema mit keinen konkreten Attributen eines existierenden Realsystems zu belasten. So wird z.B. für die Haltestellen das nur unbedingt notwendigste und zwar Name und Koordinaten gespeichert. Hier könnte man sich vorstellen, dass weiter Attribute von Interesse wären wie etwa Barrierefreiheit oder weitere Ausstattung der Haltestelle. Für die Strecken selbst könnte man sich noch viel mehr Attribute vorstellen wie etwa Art des Fahrzeugs, Fahrplan an Feiertagen, Fahrkosten usw.

Möchte man etwa für die gesamte Verbundlinie eines Landes Oeffitrack erfolgreich einführen und sämtliche Informationen dieses Verkehrsunternehmens abbilden müsste man das Datenbankschema entsprechend erweitern.

10. Installation und Anmerkungen

Eine Installationsanleitung liegt separat bei. Allgemein ist das entwickelte System Oeffitrack eine klassische PHP-MySQL Anwendung und im speziellen eine CodeIgniter-Applikation. Dementsprechend einfach gestaltet sich die Installation des Systems.
Ein weiterer Vorteil ist, dass ich mich dazu entschlossen habe das Grundsystem der Allgemeinheit als OpenSource auf Github (https://github.com/chm0815/oeffitrack) freizugeben.
Möchte man ein reales System etwa für die österreichische Verbundlinie mit einer Vielzahl an Linien abbilden müsste man sicher Schnittstellen zum Import/Export dazu programmieren. Für kleinere Events mit Shuttleservice (etwa Festivals oder Messen) könnte das System aber bestimmt ohne große zusätzliche Entwicklungs- und Installationskosten in Betrieb gesetzt werden.