Todo list

Bell

Thema: Webanwendung zum Finden eines optimalen Standortes in Abhängigkeit vom Weg mithilfe von Online-Kartendiensten

Gernot Zacharias

2. April 2019

1 Inhaltsverzeichnis

2 Einleitung

Durch den aufstrebenden Online-Versand-Handel, wie zum Beispiel durch Onlineversandapotheken, entstehen neue Probleme. Die Versandhändler benötigen Lagerhäuser für ihre Waren und die Fahrtwege vom Lager zum Kunden sollten möglichst minimal gehalten werden, um Zeit und Geld zu sparen. Durch die Verkürzung der Fahrwege wird auch die Gesundheit und die Natur weniger belastet. Auch für Privatanwender kann es interessant sein, die Fahrtwege zu minimieren, um z.B. den Weg auf Arbeit kurz zu halten, oder um einen Wohnort mit niedrigen Mieten zu wählen. Somit lässt sich auch der Treibstoff verbrauch reduzieren, was dem Geldbeutel und der Umwelt zu Gute kommt. Dieses Problem lässt sich durch eine Anwendung lösen, die aus der Häufigkeit der angesteuerten Orte eine optimale Position bestimmt mit möglichst kurzen Fahrwegen. Diese Anwendung kann zum Beispiel auf Basis der Kartendienste von 'Google Maps' und 'Open Street Map' realisiert werden. Ich habe mich letztendlich für eine Webanwendung entschieden, da Webanwendungen meistens plattformunabhängig sind. Somit ist keine Installation von zusätzlicher Software, außer einem aktuellen Webbrowser, nötig ist. Des weiteren habe ich mich für den freien Kartendienst von ,,Open Street Map„ entschieden, da dieser die meiste Freiheit bietet. Auch gibt es für ,,Open Street Maps“ eine größere Auswahl an Nutzerschnittstellen. Die Anwendung funktioniert sehr gut. Es gab auch positives Feedback von anderen Personen, die das Programm getestet haben. Allerdings bietet die Anwendung auch noch viel Spielraum für individuelle Erweiterungen, Verbesserungen und Änderungen.

„Webanwendungen werden immer beliebter“ -¿ „deshalb habe ich auch eine erstellt“ ist für mich kein gutes Argument. Besser: 1. Was ist das Problem? (Fahrtwege, Mieten) 2. Wie soll das Problem gelöst werden? (Anwendung) 3. Details (Webanwendungen werden beliebter wegen folgenden Vorteilen (Quelle), diese vorteile treffen hier auch zu

2.1 Problemstellung

Der Online Versandhandel ist ein zurzeit stark wachsender Sektor der Wirtschaft. Deshalb verwundert es auch nicht das es heutzutage auch schon Online-Versand-Apotheken gibt, welche einem direkt die Medikamente an die Tür liefern. Dies ist vor allem für bewegungseingeschränkte Personen interessant, da diese sich nun nicht mehr zu nächsten Apotheke bewegen müssen. Auch ermöglicht es Personen die nicht in der Nähe einer Apotheke wohnen, sich die benötigten Medikamente zu besorgen. Für die Anbieter dieser Versandapotheken ist es nun interressant herauszufinden, wie man Zeit und Geld beim Versand einsparen kann. Dazu kann man zum Beispiel die Warenlager dort plazieren, wo die Nachfrage am größten ist.

2.2 Grundlagen

2.2.1 World Wide Web

(Geschichte WWW) In der heutigen Zeit gewinnt das World Wide Web (WWW) immer mehr an Bedeutung und ist fast schon nicht mehr wegzudenken. Das World Wide Web ist der Teil des Internets, in dem die Daten mithilfe des HTTP/HTTPS -Protokolls ausgetauscht werden, wobei ein Computer, der Server mit der entsprechenden Webseite, die Daten auf Anfrage des Clients zu diesem schickt. Das WWW umfasst also fast alle öffentlich zugänglichen Webseiten. Das World Wide Web (Berners-Lee u. a., 1992)

2.2.2 HTML

Die Hyper Text Markup Language (HTML) ist die benutzte Sprache im WWW und beschreibt eine Textdatei, in welcher mit einer bestimmten Syntax geschrieben werden muss. Diese Textdatei wird dann von einem Browser analysiert und zeigt dem Endnutzer den Text mit entsprechender Formatierung. Reine HTML Webseiten besitzten keine dynamischen Elemente und bieten dem Nutzer wenig Interaktionsmöglichkeiten.

2.2.3 JavaScript

JavaScript ist eine Scriptsprache welche direkt in eine HTML Datei implementiert werden kann und auch vom Browser entsprechend verarbeitet wird und dynamische Webseiten erlaubt. JavaScript wird über den ¡script¿...¡/script¿ -tag in die HTML Datei eingebunden.

2.2.4 Google Maps

Google Maps ist der bekannste Online Kartendienst und bietet viele Möglickeiten zum erstellen von dynamischen Kartendarstellungen auf der eigenen Webseite. Allerdings benötigt man zur Nutzung der Google Maps API einen benutzerbezogenen Key, welchen man allerdings nur gegen Angabe von Kreditkarten Informationen erstellen kann.

2.2.5 Open Street Map

Open Street Map spielt im Vergleich mit Google Maps eine untergeordnete Rolle. Dafür bietet Open Street Map freies Kartenmaterial an und es steht eine große Menge an freien API's zu Verfügung. In meinem Prototypen setze ich auf Leavelet, welches die Kartendaten über eine externe Webseite bezieht, welche das Kartenmaterial von Open Street Map als .png zur Verfügung stellt.

2.2.6 Koordinatenberechnung

Koordinaten Längengrad und Breitengrad:
wgs84 ?
https://www.linz.govt.nz/data/geodetic-system/datums-projections-and-heights/geodetic-datums/world-geodetic-system-1984-wgs84
https://www.kompf.de/gps/distcalc.html

3 Existierende Lösungen

WIGeoGIS[1]: - Webgis - WIGeoATP QGIS - zu umfangreich, für Laien undurchsichtig - keine Webanwendung was gibt es für Quellen oder ähnliche Implementierungen

[1] `https://www.wigeogis.com`[02.04.2019 10:58:22]

4 Zielstellung

Mein Ziel ist es eine Anwendung zu kreiren, welche aus der Häufigkeit des Ansteuerns von bestimmten Orten, einen Ort bestimmt, von welchem aus die Gesamtdistanz,in Abhängigkeit von der Häufigkei, zu den gegebenen Orten möglichst klein ist.

Gesucht ist eine Anwendung, die diesen kürzesten Gesamtweg (oder die Zeit) minimiert. Wenn JavaScript: Interface mit Google Maps, keine Installation nötig, Webanwendungen auf dem Vormarsch... Wenn andere Programmiersprache, auch da kurz schreiben warum. (zum Beispiel in Schule beigebracht)

5 Algorithmus

$$\lambda = \left(\sum_{i=1}^{n} \lambda_i \cdot w_i \right) \cdot \left(\sum_{i=1}^{n} w_i \right)^{-1} \tag{5.1}$$

$$\varphi = \left(\sum_{i=1}^{n} \varphi_i \cdot w_i \right) \cdot \left(\sum_{i=1}^{n} w_i \right)^{-1} \tag{5.2}$$

$$\tag{5.3}$$

Die Anwendung berechnet einen gewichteten Mittelpunkt anhand der vom Nutzer angegebenen Priorität. Wärend des Setzens der Marker wird bereits ein Mittelpunkt berechnet. Durch einen Klick auf den Marker erscheint ein Popup in welchem man die Priorität ändern kann. Danach wird die Position des Mittelpunktes erneut berechnet. Der Mittelpunkt wird durch drei Kreise dargestellt, einen inneren roten, einen mittleren gelben und durch einen grünen äußeren Kreis.

6 Implementierung

Für die Kartendarstellung wird Leaflet (Crickard III, 2014), eine OpenSource Bibliothek für die Darstellung von Kartenmaterial aus verschiedenen Quellen, genutzt. Eine Kartendatenquelle, die diese Anwendung benutzt, ist OpenStreetMap[1]. Für die Webanwendung ist nur ein halbwegs aktueller Webrowser notwendig. Die Webanwendung besteht im wesentlichen bloß aus einem HTML Dokument und einer Javascript Datei sowie der Leaflet Bibliothek.

```
...
<link rel="stylesheet" href="leaflet/leaflet.css" crossorigin=""/>
<script src="leaflet/leaflet.js"></script>
...
<div id="map"></div>
<script src="js/leafscript1.js" type = "text/javascript"></script>
...
```

Das Script *leaflet1.js* erstellt eine neue Variable *mymap*, die dann als Objekt für die Kartendaten dient. Die Karte wird dabei erstellt und hat als Mittlepunkt die Stadt Leipzig(Längengrad: 51.339°Nord, Breitengrad: 12.381°Ost,). Die Ansicht ist standartmäßig auf eine Zoomstufe von 12 eingestellt, da man so die gesamte Stadt und das Umland sehen kann. Als nächstes wird dann zur Karte eine neue Ebene hinzugefügt, welche die Bildinformationen von einem OpenStreetMap Server abfragt und darstellt. Außerdem wird festgelegt wie weit man in die Karte rein- und rauszoomen kann.

```
const mymap = L.map('map').setView([51.33918, 12.38105], 12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png?lang=de', {
   maxZoom: 20,
   minZoom: 5,
   attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a>
      contributors',
}).addTo(mymap);
```

Als nächstes wird eine Varible für einen Marker initialisiert, sowie eine Konstante i und ein Feld für die gesetzten Marker. Auch werden Variablen für die Kreise, die den Mittelpunkt anzeigen, definiert.

```
let marker;
const i=0;
let markers = [];
```

[1] https://www.openstreetmap.org

```
let kreis1;
let kreis2;
let kreis3;
```

Damit der Nutzer Marker auf die Karte setzen kann wird einen neue Funktion *onMapClick(e)* definiert, die, an der angeklickten Position, auf der Karte, einen Marker setzt. Jeder Marker bekommt eine ID zugewiesen, damit man ihn später wieder entfernen kann. Auch wird dem Marker standardmäßig eine Priorität mit dem Wert 5 zugewiesen. Die ID wird aus der Länge des *markers* Feld definiert. Dem Marker wird auch ein Popup zugewiesen. Dieses Popup zeigt die Marker ID, Priorität und einene Button zum entfernen des Markers. Das Popup enthält auch ein Eingabefeld zum ändern der Priorität. Beim Ändern der Priorität wird die Funktion *change_marker()* aufgerufen und übergibt die ID des Markers und die Priorität. Der Button ruft bei einem Klick die Funktion *clear_marker()* auf. Danach wird zur Karte ein neuer Layer hinzugefügt und die Marker die im Array *Markers* enthalten sind, werden dem Layer hinzugefügt. Als letztes wird noch die Funktion *center()* aufgerufen.

```
function onMapClick(e){
   let id;
   if (markers.length < 1) {
      id = 0;
   }else {
      id = markers[markers.length - 1]._id + 1;
   }
   marker = new L.marker(e.latlng, {draggable:false}).addTo(mymap);
   marker._id = id;
   marker._prio = 5;
   marker.bindPopup('<b>Marker '
      + marker._id
      +'</b><br>'
      + 'Priorität:'
      + '<br><input type="number" value="'
      + marker._prio
      + '" oninput="change_marker('
      + marker._id
      +', this.value)" placeholder="Priorität" min="0" max="1000" />'
      + '<br>==========================<br>'
      + '<input type="button" value="Entferne Marker" onclick="clear_marker('
      + marker._id
      + ')" />'
   );
   mymap.addLayer(marker);
   markers.push(marker);
   center();
}
```

Die nächste wichtige Funktion ist *center()*, da diese für die Berechnung des Mittlepunktes zuständig ist. Der Mittelpunkt wird somit immer bei einer Änderung eines Markers neu berechnet. Zuerst wird geprüft ob bereits die Kreise eins(Grüner Kreis), zwei(Gelber Kreis) und drei(Roter Kreis) vorhanden sind und entfernt diese wenn sie bereits vorhanden sind.

```
function center(){
   if(kreis1&&kreis2&&kreis3){
      mymap.removeLayer(kreis1);
      mymap.removeLayer(kreis2);
      mymap.removeLayer(kreis3);
   }
   ...
}
```

Darauffolgend wird eine Funktion *add* definiert, welche zwei Werte addiert. Danach wird die Summe aus allen Prioritäten der Marker gebildet. Als nächstes wird die Summe der Längen- und Breitengrade addiert und jeweils durch die Summe der Prioritäten geteilt.

```
   ...
   const add = (a,b)=>a+b;
   const prioSum = markers.map(m => m._prio).reduce(add,0);
   const lat = markers.map(m => m._latlng.lat *
       m._prio).reduce(add,0)/prioSum;
   const lng = markers.map(m => m._latlng.lng *
       m._prio).reduce(add,0)/prioSum;
   ...
```

Der Mittelpunkt wird durch drei Kreise, mit jeweils anderen Farben, dargestellt. Der erste Kreis hat die Farbe grün und stellt den äußeren Kreis dar. Kreis zwei stellt den mittleren Bereich dar und hat die Farbe gelb und Kreis drei hat den kleinsten Radius und besitzt die Farbe rot. Die Kreise haben besitzen den gewichteten Mittelpunkt als Mittelpunkt.

```
   ...
   kreis1 = L.circle([lat, lng], {
      color: 'green',
      fillColor: '#00ff00',
      fillOpacity: 0.2,
      radius: 1000,
   }).addTo(mymap);
   kreis2 = L.circle([lat, lng], {
      color: 'yellow',
      fillColor: '#ffff00',
      fillOpacity: 0.3,
      radius: 500,
   }).addTo(mymap);
```

```
kreis3 = L.circle([lat, lng], {
   color: 'red',
   fillColor: '#f03',
   fillOpacity: 0.5,
   radius: 100,
}).addTo(mymap);
...
```

Wenn man nun Marker wieder entfernen möchte Braucht man eine neue Funktion *clear_marker*, welche die gesetzen Marker wieder entfernt.

Hier werden dann die technischen Details der Implementierung und gegebenenfalls der Algorithmen beschrieben. Verwendete Technologien, Lizenz, Systemvoraussetzungen und so weiter.; Aufpassen: Fehlerquelle sinus/cosinus Bogenmaß oder Gradmaß;

7 Schlussfolgerung

7.1 Auswertung

wieviel bringt das wirklich

7.2 Ausblick

Momentan berechnet die Anwendung die Entfernungen nur per Luftlinie und lässt Straßen und natürliche Hindernisse, wie zum Beispiel Gewässer und Wälder, außer Acht. Auch basiert die Berechnung bisher nur auf Linien in der Ebene, aber die Erde ist ein drei dimensionales Objekt und der Abstand zwischen zwei Punkten müsste durch einen Kreisbogen beschrieben werden. Somit besteht noch viel Spielraum für die Erweiterung der Anwendung, zum Beispiel Integration von einem Wegfindealgorithmus, einer veränderten Distanzrechnung, ... Zeit, statt Distanz; nicht Luftlinie, sondern kürzesten Weg (z.B. über Google Maps API);global, statt Leipzig; Maximallänge für Wege - Erweiterung um Speicherung der gesetzten Punkte, Einlesen von vorgebebenen Daten(Cookies, LocalStorage, Get-Parameter, ...)

Für ähnliche Anwendungsfälle Mehrere Kriterien für standortwahl - Verfügbarkeit von Arbeitskräften - Standortkosten - Prozessfluss -

7.3 Abkürzungsverzeichnis

API	Application programming Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JS	JavaScript
PNG	Portable Network Graphic
WGS84	52
WWW	World Wide Web

Literaturverzeichnis

[Berners-Lee u. a. 1992] Berners-Lee, Tim ; Cailliau, Robert ; Groff, Jean-François ; Pollermann, Bernd: World-Wide Web: The information universe. In: *Internet Research* 2 (1992), Nr. 1, S. 52–58

[Crickard III 2014] Crickard III, Paul: *Leaflet. js Essentials.* Packt Publishing Ltd, 2014