

994b.png

Achtung:

Wenn auf Ihrer Karte der Layer `dwd:Warnungen_Gemeinden_vereinigt` nicht angezeigt wird, kann es auch daran liegen, dass es zur Zeit keine Warnungen gibt. Dieser Layer enthält nur Daten, wenn aktuell Wetterwarnungen vorliegen. Die grünen Polygone, die die Landkreise darstellen, im Layer `dwd:Warngebiete_Kreise` werden dagegen immer eingeblendet.

In diesem Kapitel haben wir ...

Sie können nun eine digitale Karte mit Leaflet in ein HTML Dokument einbinden, Sie wissen was geografische Koordinaten sind und wie die Imagedateien für digitale Karten erstellt werden. Darauf aufbauend können wir nun im nächsten Kapitel Ihre digitale Karte mit Geodaten füllen.

Die Karte mit Daten bestücken

Bisher haben wir ausschließlich vollständige Layer zur Karte hinzugefügt. Wie das Beispiel mit den Warnhinweisen vom Deutschen Wetterdienst zeigt, können diese Layer dynamisch mit Daten bestückt werden.

In diesem Kapitel werden wir ...

In diesem Kapitel zeige ich Ihnen nun, wie Sie selbst Layer mit Daten erstellen und als Schicht auf Ihrer Karte anzeigen können. Dabei verwenden wir

einfache geometrischen Objekten, die in einem zweidimensionalen Raum definiert sind. Konkret sind die Punkte, Linien und Polygone.

- **Punkte/Marker:**

Das einfachste Objekt ist ein Punkt. Auf einer Landkarte wird ein Punkt mit Zahlen – den Koordinaten – beschrieben. In Leaflet gibt neben dem Punkt noch ein komfortableres Element – nämlich den Marker. Auf einer realen Karte könnte ein Marker zum Beispiel einen interessanten Ort, also einen Point of Interest (POI) darstellen.

- **Linien:**

Eine Linie besteht aus mindestens zwei Punkten, die miteinander verbunden sind.

- **Polygone:**

Ähnlich wie eine Linie besteht ein Polygon aus mehreren miteinander verbundenen – zusätzlich ist beim Polygon auch der erste Punkt mit dem letzten Punkt verbunden. Somit stellt ein Polygon eine geschlossene Figur dar.

Wie können Sie nun Punkte, Marker, Linien und Polygone auf Ihre Karte zeichnen? Das ist Thema dieses Kapitels. Und auch, wie und warum Sie die Objekte in Layer-Gruppen und Feature-Gruppen gruppieren können oder sollten.

Zusätzlich sehen wir uns an, wie Sie die Karte mobil passend anzeigen und wie Sie auf Ereignisse reagieren können.

Punkte, Marker, Linien, Polygone, Rechtecke und Kreise als Leaflet Objekte

Leaflet bietet Ihnen jede Menge verschiedene Objekte. Beginnen wir mit den einfachsten – den Punkten und Markern.

Punkte und Marker

Ich gehen wir noch einmal von unserer Basiskarte aus. Zur besseren Übersicht habe ich Ihnen das HTML-Grundgerüst dazu nachfolgend noch einmal abgedruckt.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
```

```

<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
</script>
</body>
</html>
<!--index_989.html-->

```

In der nächsten Abbildung sehen Sie der Vollständigkeit halber auch noch einmal die Karte, die angezeigt werden sollte, wenn Sie die vorhergehende HTML-Datei in einem Browser öffnen.



993.png

Diese Karte ist bisher nicht sehr spannend. Landkarten, die Straßen und Orte anzeigen findet man wie Sand am Meer. Auf Ihrer Website geht es sicher um etwas Besonderes und so möchten Sie sicher auch einen besonderen Ort hervorheben oder ein besonderes Thema beschreiben. Beginnen wir mit einem besonderen Ort – beziehungsweise einem besonderen Punkt.

Punkte

Die Leaflet [Klasse Point](#) stellt einen Punkt mit X- und Y-Koordinaten in Pixeln dar. Die Koordinaten werden in Leaflet im Dezimalformat formatiert. Die erste Zahl steht für die Latitude – als die geografische Breite – und die zweite Zahl für die Longitude – also die geografische Länge.

Achtung:

Sie haben richtig gelesen. Bei einem Punkt arbeite Leaflet nicht mit geografischen Koordinaten. Hier müssen Sie tatsächlich die Pixel der Grafik, mit die Karte angezeigt wird, angeben.

Objekte vom Typ Point sind in Leaflet nicht zur Anzeige gedacht. Vielmehr wird mit Ihnen gearbeitet. Zum Beispiel gibt es die [Methode panBy\(<Point> offset, <Pan options> options?\)](#) mit der die Karte um eine gegebene Anzahl von Pixeln verschoben werden kann.

Sie merken schon. Ein Objekt vom Typ Point ist nicht das, was wir möchten, wenn wir einen Punkt auf der Karte hervorheben möchten. Um einen Punkt als Besonders zu markieren, möchten Sie sicher eher einen Marker mit Informationen an dieser Stelle – also an den geografischen Koordinaten – anzeigen. Und für diesen Zweck bietet Leaflet ein spezielles Objekt – das Marker Objekt.

Im nachfolgenden Programmcode-Beispiel sehen wir uns aber zunächst kurz an, was Sie mit einem Punkt anstellen können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
```

```
var point = L.point(400, 600);  
mymap.panBy(point);  
</script>  
</body>  
</html>  
<!--index_988.html-->
```

Wenn Sie das vorhergehende HTML-Dokument in Ihrem Browser öffnen, müssen Sie ganz schnell gucken. Nur dann können Sie erkennen, dass die Karte unmittelbar nach dem Öffnen um 400 Pixel nach rechts und 600 Pixel nach unten geschoben wird.

Hinweis:
Anstelle von

```
var point = L.point(400, 600);  
mymap.panBy(point);
```

hätten Sie auch ganz einfach

```
mymap.panBy([400, 600]);
```

oder

```
mymap.panBy(L.point(400, 600));
```

schreiben können.

An dieser Stelle füge ich kein Bild, auf dem das Ergebnis dargestellt ist, ein. Das Verschieben der Karte wäre auf diesem nicht zu erkennen.

Marker

Die Leaflet [Klasse Marker](#) wird verwendet, um anklickbare und/oder verschiebbare Symbole auf einer Karte anzuzeigen. Die Klasse erweitert die Klasse [Layer](#). In der Regel wird dieses Symbol mit einem Pop-up erweitert, auf dem weitere Informationen zu der entsprechenden Stelle auf der Erde enthalten sind.

Machen wir mit einem einfachen Beispiel weiter und fügen einen einfachen Marker zu unserer bisher noch langweiligen Karte hinzu.

```
<!DOCTYPE HTML>
```

```

<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet"
      href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css" />
<script
      src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var myMarker = L.marker([50.27264, 7.26469],
{
    title:"Gering",
    alt:"Ein schönes kleines Dorf auf dem Maifeld in der Eifel",
    draggable:true
}
).addTo(mymap);

</script>
</body>
</html>
<!--index_987.html-->

```

Was haben wir genau gemacht? Mit dem Text `var myMarker = L.marker([50.27264, 7.26469])` haben wir ein Marker Objekt, das an den Koordinaten [50.27264, 7.26469] angezeigt wird, instanziert. Im Weiteren haben wir die Optionen `title`, `alt` und `draggable`. Dabei setzen Sie mit `title` den Tooltip der erscheint, wenn Sie die Maus über den Marker bewegen, mit `alt` setzen Sie den Text für das `alt` Attribut des Marker Bildes und mit `draggable` legen Sie fest, ob der Marker mithilfe der Maus auf der Karte bewegt werden kann – oder nicht.

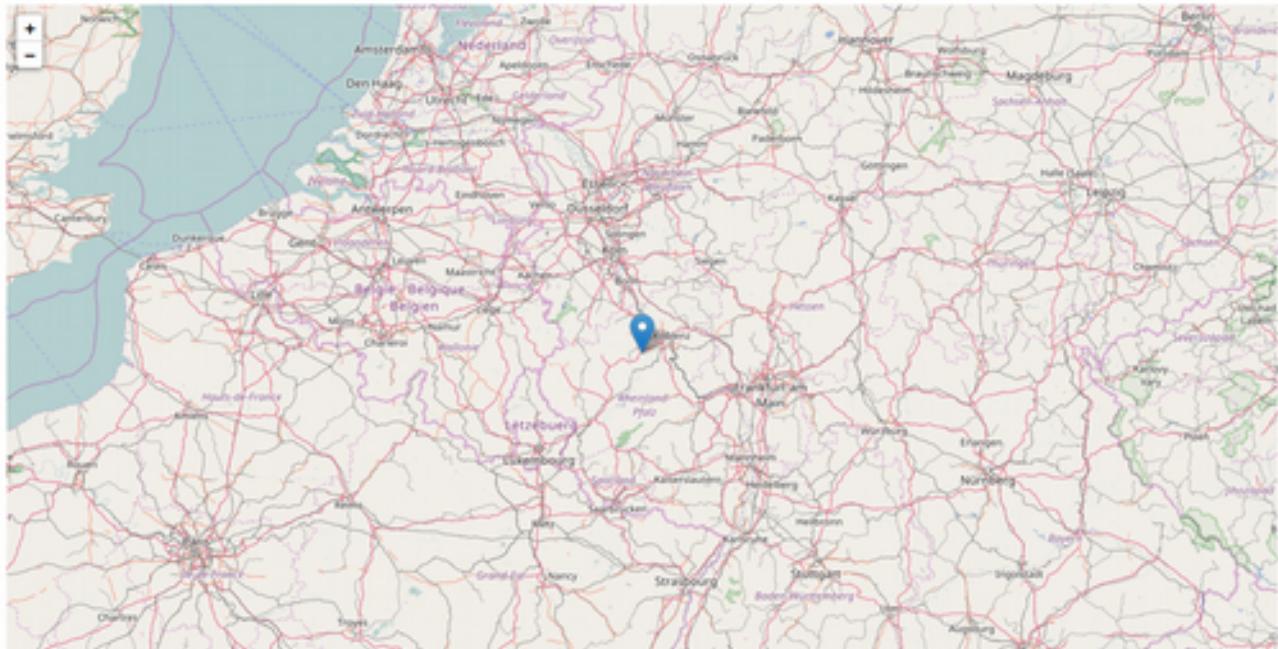
Das Image-Tag des Markers fügt Leaflet wie folgt ins HTML-Dokument ein:

```

```

Alle Optionen und Methoden, die Ihnen die Leaflet Klasse Marker bietet, können Sie in der [Leaflet Dokumentation](#) nachlesen.

Im nächsten Bild sehen Sie den Marker in die Karte integriert. Als Bild wurde das [Standardbild](#) verwendet, da die Option icon nicht gesetzt wurde. Wie Sie ein benutzerdefiniertes Icon verwenden können, sehen wir uns in Kapitel *Custom Markers* genauer an. Den Marker können Sie im Browser mit der Maus bewegen, da die Option draggable auf true gesetzt wurde. Probieren Sie es aus!



992.png

Hinweis:

Sie müssen für einen Marker keine Variable instanziieren. Sie können den Marker auch ganz einfach so zur Karte hinzufügen:

```
L.marker([50.27264, 7.26469], { title:"Gering", alt:"Ein schönes kleines Dorf auf dem Maifeld in der Eifel", draggable:true }).addTo(mymap);
```

Beachten Sie dabei aber folgendes: Falls Sie den Marker später einmal modifizieren möchten, benötigen Sie einen Namen. Andernfalls können Sie

den Marker nicht identifizieren und somit auch nicht zum Ändern auswählen.

Objekte, die aus mehr als einem Punkte bestehen

Linien

Linien können Sie in Leaflet mit der Klasse [Polyline](#) erstellen. Die Klasse `Polyline` erweitert die abstrakte Klasse [Path](#). Diese Klasse ermöglicht es Ihnen eine einfache Linie oder mehrere aneinandergereihte Liniensegmente zu erstellen.

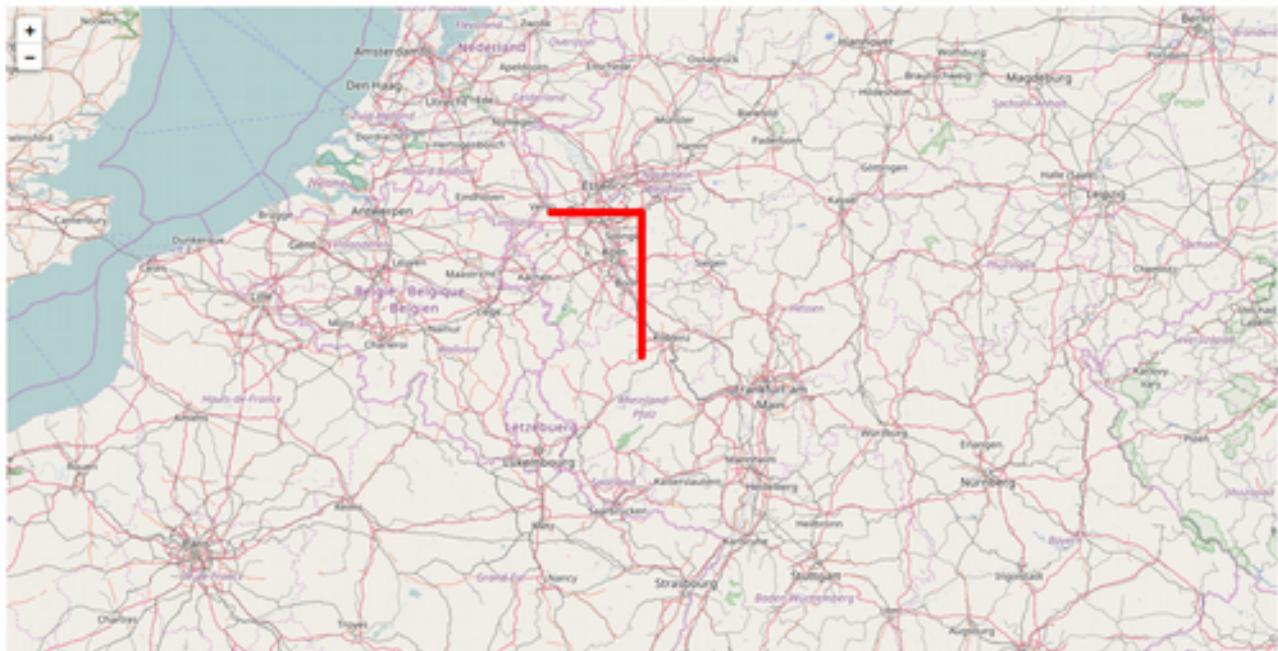
Im nachfolgenden Programmcodebeispiel habe ich den Text, der für die Erstellung des `Polyline` Objektes verantwortlich ist, fett formatiert.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
{color: 'red', weight:8}).addTo(mymap);
</script>
</body>
</html>
<!--index_986.html-->
```

986.html

Im Beispiel wird ein `Polyline` Objekt instanziert, das aus drei Punkten besteht. Außerdem werden die Optionen `color` und `weight` verwendet. Diese Optionen erbt die Klasse `Polyline` von der Klasse `Path`. Konkret legen wir im Beispiel mit `color = 'red'` die Farbe der Linie fest – diese soll rot sein. Mit `weight:8` bestimmen wir, dass die Linie 8 Pixel dick sein soll.

In der nächsten Abbildung sehen Sie genau diese Linie.



991.png

Polygone

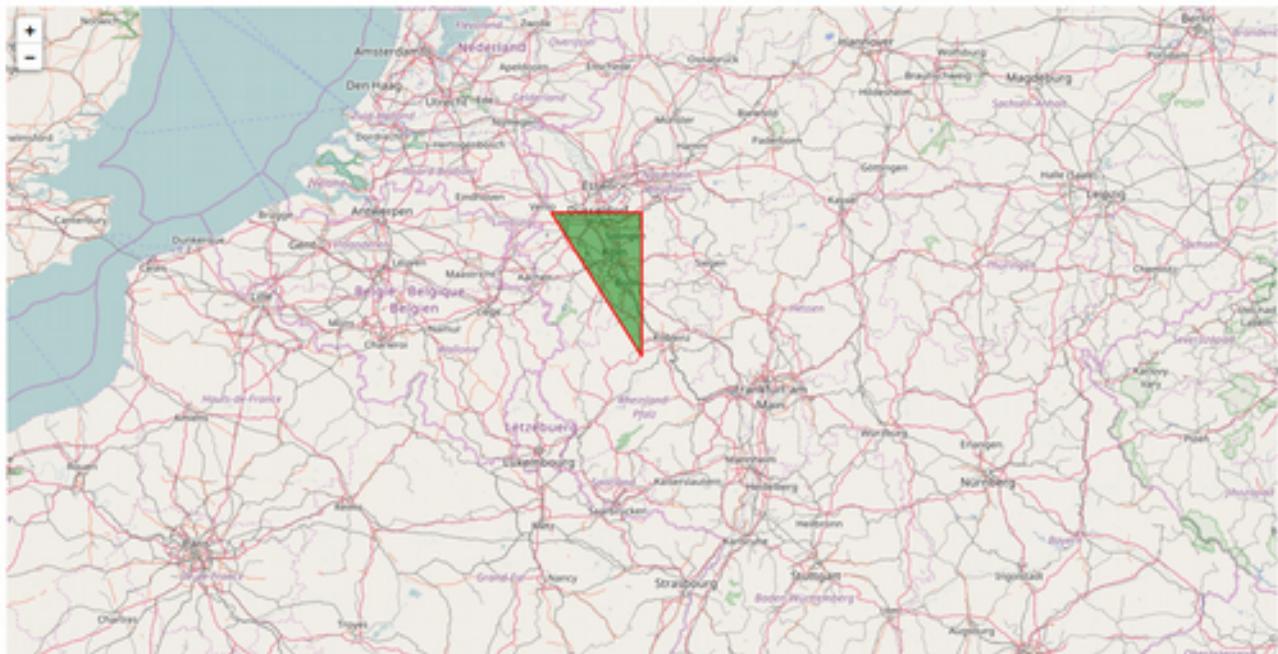
Ein `Polygone` ist eine geschlossene Linie – also eine Linie die einen Innenbereich von einem Außenbereich abtrennt. Mit der Klasse `Polygone` können Sie, wie der Name schon sagt, `Polygone` Objekte auf Ihre Karte zeichnen. Die Klasse `Polygone` erweiterte die Klasse `Polyline` und somit auch die Klasse `Path`. Alle Optionen dieser Klasse können Sie also auch mit einem `Polygone` Objekt nutzen.

Achtung:

Die Punkte, die Sie beim Erstellen eines Polygons übergeben, sollten als letzten Punkt keinen Punkt enthalten, der dem ersten entspricht. Es ist besser, solche Punkte herauszufiltern. Leaflet schließt Ihr Polygon automatisch und, wenn der erste Punkt mit dem letzten Punkt übereinstimmt, kann dies zu Problemen führen.

Wie Sie ein Polygon in Leaflet erstellen, zeigt Ihnen der nachfolgende Programmcode und das konkrete Polygon sehen im darauf folgenden Bild. Alle Methoden und Optionen, die Sie auf ein Polygon Objekt anwenden können, finden Sie in der Leaflet Dokumentation.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var polygon = L.polygon([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
{color: 'red', weight:2, fillColor:'green',
fillOpacity:0.5}).addTo(mymap);
</script>
</body>
</html>
<!--index_985.html-->
```



990.png

Rechtecke und Kreise

Um einen Kreis oder ein Rechteck zu erstellen benötigt man eigentlich keine eigene Klasse. Man könnten diese mit der Klasse Polygon erzeugen. Beim Kreis wäre dies allerdings sehr mühselig und auch für ein Rechteck gibt es einfacherer Verfahrensweisen. Leaflet bietet deshalb für diese beiden Formen spezielle Klassen an.

Das Zeichnen von Rechtecken und Kreisen ist kein Hexenwerk. Der Vollständigkeit halber finden Sie aber in den nächsten beiden Kapiteln jeweils ein Beispiel für beide Formen.

Rechtecke

Der nachfolgende Programmcode zeigt Ihnen an einem Beispiel, wie Sie ein Rechteck – also ein Objekt vom Typ [Rectangle](#) – in eine Leaflet Karte einfügen.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
```

```

<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var myRectangle = L.rectangle([
[50.27264, 7.26469],
[51.27264, 6.26469]
],
{color: "yellow", weight: 8, fillColor:"purple"}).addTo(mymap);
</script>
</body>
</html>
<!--index_984.html-->
```

Als Parameter geben Sie für das Rechteck immer die diagonal versetzten beiden Ecken an. Dabei ist es egal, in welcher Reihenfolge Sie diese angeben.

```

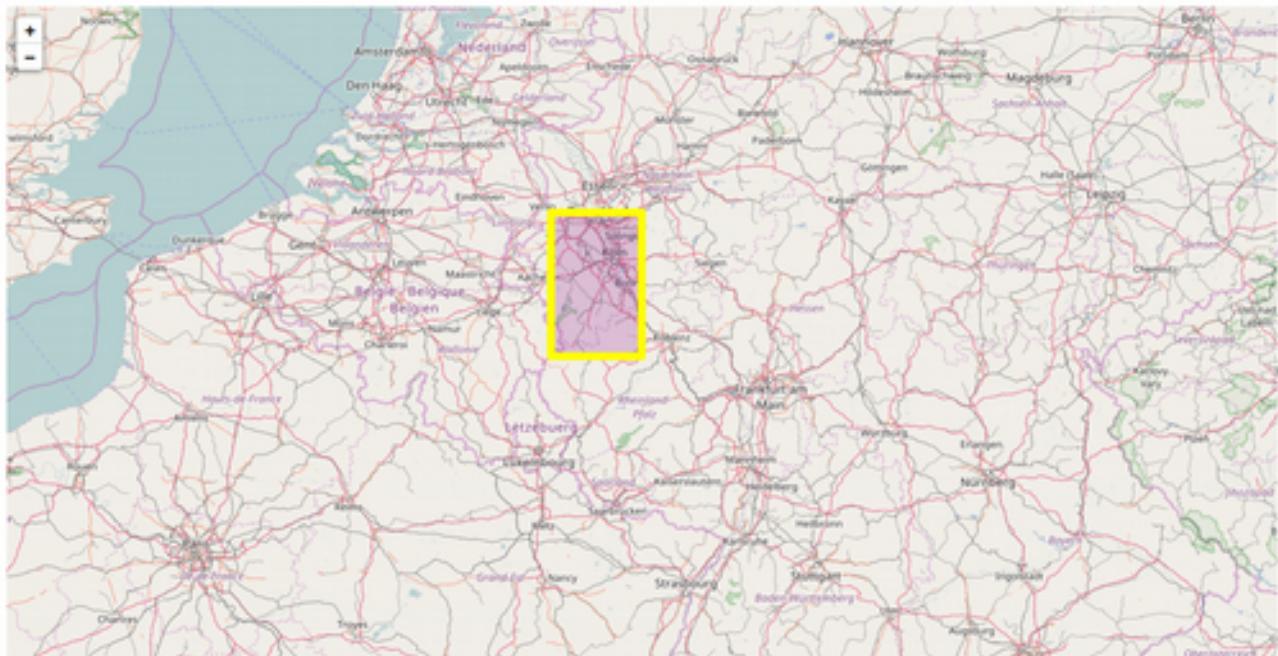
var myRectangle = L.rectangle([
[50.27264, 7.26469],
[51.27264, 6.26469]
])
```

sieht auf der Karte genauso aus, wie

```

var myRectangle = L.rectangle([
[51.27264, 6.26469],
[50.27264, 7.26469]
]).
```

Und wie das Rechteck aussieht, zeigt Ihnen die nachfolgende Abbildung.



989.png

In diesem Zusammenhang ist vielleicht ganz interessant wie Leaflet das Koordinatensystem der Erde verarbeitet. Dieses Koordinatensystem ist ja kein gewöhnliches Koordinatensystem, sondern ein sphärisches. Unter einer Sphäre versteht man in der Mathematik ganz vereinfacht ausgedrückt die Oberfläche einer Kugel. Und auf einer Kugel führen, im Gegensatz zu einer Ebene, immer zwei Linien auf direktem Wege zu einem anderen Punkt. Die nachfolgenden Abbildungen zeigen Ihnen, dass Leaflet dieses Problem vereinfacht. Es behandelt das sphärische Koordinatensystem als normales Koordinatensystem. Um von dem Punkt [50, -180] zum Punkt [51, 180] zu gelangen, muss man mit Leaflet einmal um die ganze Erde laufen.



929a.png



Kreise

Anhand des nachfolgenden Programmcode können Sie erkennen, wie ein Kreis – also ein Objekt vom Typ [Circle](#) – in eine Leaflet Karte eingefügt wird.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

L.circle(
[50.27264, 7.26469],
100000,
{color: "red", weight: 8, fillColor:"green"}).addTo(mymap);
</script>
</body>
</html>
<!--index_983.html-->
```

Ein Kreis wird anhand seines Mittelpunktes und seines Radius definiert. Der Radius muss dabei in Metern angegeben. In den Beispielen habe ich den Rand

meist mit 8 Pixel festgelegt. Diese Breite passt auch für die Zoom-Stufe 7. Probieren Sie doch einmal aus, wie der Rand aussieht, wenn Sie die Karte vergrößern. Sie werden feststellen, dass man irgendwann nur noch den Rand sieht und es wichtig sein könnte, dessen Größe immer relativ zum Zoom zu setzen.



988.png

Mehrere Poly- Objekte auf einer Ebene

In den vorhergehenden Beispielen haben wir jedes Element auf einem separaten Layer – also einer separaten Ebenen – abgebildet. Spätestens, wenn Sie selbst unterschiedliche Geodaten auf einer Karte darstellen möchten werden Sie sich wünschen, Objekte mit gleichen Eigenschaften auf einem Layer gruppieren zu können. Denn nur so können sie alle Elemente gleichzeitig ansprechen. Stellen Sie sich vor, Sie möchten auf Ihrer Website Touren für Aktivurlauber beschreiben. Diese Touren sind unterteilt in Wanderer, Bergsteiger, Gipfelstürmer und Freikletterer. Auf der Karte kann ein Kunde Angebote für eine bestimmte Region heraus filtern. Wenn Sie pro Touren-Typen einen [Layer](#) anlegen und die Touren entsprechend ihres Typs auf diesen Layern ablegen, können Sie es dem Kunden zusätzlich auf einfache Art ermöglichen, nur die für ihn relevanten Touren einzublenden. Warum erkläre ich Ihnen dies an dieser Stelle? Alle `Polyline` Objekte alle `Polygone` Objekte zusammen definiert wurden, liegen automatisch zusammen auf einem eigenen separaten Layer.

Das Setzen von Klammern beim Arbeiten mit `Polygon` Objekten und `Polyline` Objekten die mehrere Objekte gleichzeitig auf einem Layer erstellen, kann sehr herausfordernd sein.

- Sie müssen zum einen alle Objekte – `Polygon` Objekte oder `Polyline` Objekte – mit einer äußeren Klammer versehen.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```
- Außerdem müssen Sie jedes einzelne Objekt – `Polyline` oder `Polygone` – mit einer Klammer umgeben.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```
- Und zuletzt werden auch die Koordinaten selbst noch eingeklammert.

```
[  
  [[50.17264, -7.26469], [49.27264, -6.26469]],  
  [[50.37264, -7.26469], [51.27264, -8.26469]]  
]
```

Mehrere Polyline Objekte auf einer Ebene

In diesem Kapitel gibt es nun ein praktisches Beispiel. Der nachfolgende Programmcodeausschnitt zeigt Ihnen, wie Sie mit dem instanziieren eines `Polyline` Objekt mehrer `Polyline` Objekte auf eine Ebene zeichnen können. Und das die beiden auf einer Ebene liegen, beweise ich Ihnen gleichzeitig auch noch.

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Eine OSM Karte mit Leaflet</title>  
<link rel="stylesheet" href="../leaflet/leaflet.css" />  
<script src="../leaflet/leaflet.js"></script>  
</head>  
<body>  
<div style="height: 700px;" id="mapid"></div>  
<script>
```

```

var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var multipolyline = L.polyline(
[
  [[50.17264, -7.26469],
  [49.27264, -7.26469],
  [49.27264, -6.26469]],
  [[50.37264, -7.26469],
  [51.27264, -7.26469],
  [51.27264, -8.26469]]
],
{color: 'black'}).addTo(mymap);
mymap.fitBounds(multipolyline.getBounds());
</script>
</body>
</html>
<!--index_981.html-->
```

Was zeigt Ihnen dieses Beispiel? Zunächst einmal können Sie erkennen, dass das Erstellen mehrerer `Polyline` Objekte zusammen möglich ist. Und das es auch einfacher ist erkennen Sie daran, dass Sie Optionen nur einmal zuweisen müssen. Beide Linien werden gleichzeitig schwarz gefärbt.

Am meisten Vorteile bringt aber die Tatsache, dass beide Linien eine Ebene darstellen. Die Methode `fitBounds()` verschiebt eine Karte an die Stelle, an der die übergebenen Koordinaten sich befinden und setzt die Zoom-Stufe so, dass die Karte für die übergebenen Koordinaten ideal angezeigt wird. Würden die beiden Linien nicht auf einer Ebenen liegen, wäre es schwere beide gleichzeitig mit maximalem Zoom anzuzeigen. Da sie aber auf einer Ebene liegen, sehen Sie wie nachfolgend im Bild ersichtlich, beide Linien an der richtigen Stelle auf der Erde.

Hinweis:

Noch einmal zur Erinnerung: Ein Objekt vom Typ `Polygon` und auch ein `Polyline` Objekt ist eine Ebene – die beiden Klassen erweitern die Leaflet Klasse `Layer`.



Abbildung 7
987.png

Mehrere Polygone Objekte auf einer Ebene

In diesem Kapitel zeige ich Ihnen ein Beispiel, bei dem mehrere Polygone zusammen erstellt werden.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var multipolygone = L.polygon(
[
[[50.17264, 7.26469],
```

```
[49.27264, 7.26469],  
[49.27264, 6.26469]],  
[[50.37264, 7.26469],  
[51.27264, 7.26469],  
[51.27264, 8.26469]]  
,  
{color: 'black'}}).addTo(mymap).bindPopup("Wir sind auf einer  
Ebene");  
</script>  
</body>  
</html>  
<!--index_982.html-->
```

In diesem Beispiel sehen Sie nun auch, wie Sie mit der Methode `bindPopup()` ein Pop-up zu einem Element hinzufügen können. Später im Buch werden Pop-up Fenster noch einmal Thema sein. An dieser Stelle habe ich die Methode gewählt, weil ich der Meinung bin, das diese hier gut passt. Es gibt viele unzusammenhängende Flächen auf der Erde, für die die gleichen Informationen zutreffend sind. Zum Beispiel haben viele Länder Kolonien an weit entfernten Stellen auf der Erde. Wenn nun alle zu einem Land gehörenden Gebiete als Polygon auf einem Layer zusammengefasst sind, dann ist es ein Leichtes, diesen allen Gebieten eines Landes das gleiche Pop-up Fenster zuzuweisen. Und wenn Sie nun die ganze Welt auf Ihrer Karte abdecken möchten, dann können Sie sich den Vorteil dieser [Layer-Technik](#) sicherlich gut vorstellen.

Auf der nächsten Abbildung sehen sie zwar keine Ländergrenzen. Der Einfachheit halber habe ich mich auf zwei Dreiecke beschränkt.



Abbildung 8

986.png

Mehrere sich überschneidende Polygon Objekte auf einer Ebene

Die Besonderheit bei einem Polygon ist, dass es einen Außen- und einen Innenbereich gibt. Das lesen Sie hier im Buch des Öfteren. Wenn Sie mehrere Polygone auf einem Layer zusammen erstellen, dann beeinflussen diese sich gegenseitig, wenn sie übereinander liegen. In der nachfolgenden Tabelle habe ich verschiedene Konstellationen mit Bild und Text aufgenommen. Anhand von Beispielen ist das Zusammenspiel der verschiedenen Bereiche meiner Meinung nach am leichtesten nachzuvollziehen.



Abbildung 9
928a.png

Beginnen wir mit einem einfachen Polygon:

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10],
  [10, 1] ]
])
```



Abbildung 10

Wenn Sie innerhalb dieses Polygons ein weiteres Polygon zeichnen, dann wird der Innenbereich des ersten Polygon um die Fläche des zweiten Polygon verringert. Das zweite Polygon wird praktisch aus dem ersten Poligone ausgeschnitten.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10],
  [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2]
  ]
])
```



Abbildung 11

In dieses zweite Polygon können Sie nun wiederum ein Polygon zeichnen. Nun wird diese Fläche wieder voll als Innenbereich gezeichnet.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10],
  [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2]
  ],
  [ [3, 3], [3, 4], [4, 4], [4, 3]
  ]
])
```

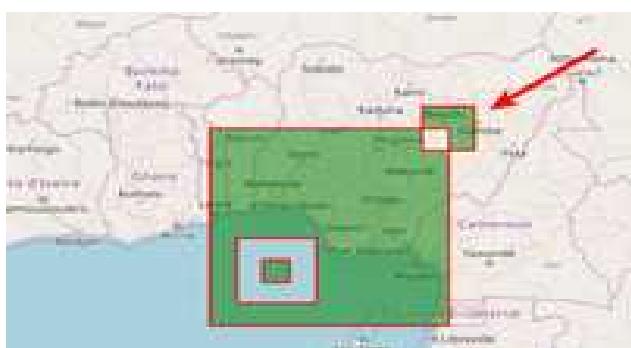


Abbildung 12
928d.png

Zeichnen Sie nun ein weiteres Polygon, dass Innenbereich und Außenbereich gleichzeitig abdeckt, dann wird der abgedeckte Innenbereich zum Außenbereich und umgekehrt.

```
L.polygon([
  [ [1, 1], [1, 10], [10, 10],
  [10, 1] ],
  [ [2, 2], [2, 5], [5, 5], [5, 2]
  ],
  [ [3, 3], [3, 4], [4, 4], [4, 3]
  ],
  [ [9, 9], [11, 9], [11, 11], [9,
  11]
  ]
])
```

Daten mit Layern gruppieren

Im vorherigen Kapitel haben Sie gesehen, dass Sie mehrere `Polygone` Objekte oder mehrere `Polyline` Objekte zusammen auf einem Layer positionieren können. Es wird aber sicher auch einmal vorkommen, dass Sie Elemente unterschiedlicher Typen auf einem Layer zusammen gruppieren möchten.

Layergruppen

Die Klasse Leaflet Klasse [LayerGroup](#) wird verwendet, um mehrere Layer oder Ebenen zu gruppieren. So können Sie diese Ebenen wie eine Ebene behandeln. Wenn Sie ein Objekt vom Typ `LayerGroup` zur Karte hinzufügen, werden alle zur Gruppe gehörenden Layer zur Karten Objekt hinzugefügt.

Der nachfolgende Programmcodeausschnitt zeigt Ihnen, wie Sie ein Objekt vom Typ `LayerGroup` erstellen und zu Ihrem Karten Objekt hinzufügen können.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var marker1 = L.marker([50.27264, 7.26469],
{
    title:"Marker1",
    alt:"Ich bin ein Marker"
})
.bindPopup('Marker1');
var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
{color: 'red', weight:8});
var myLayerGroup = L.layerGroup([marker1, polyline]).addTo(mymap);
</script>
</body>
```

```
</html>  
<!--index_980.html-->
```

Im vorherigen Beispiel habe ich einen Marker und ein Objekt vom Typ `Polyline` erstellt und beide zusammen auf dem Layer `myLayerGroup` gruppiert.

Hinweis:

Sie können auch später noch Objekt zum `LayerGroup` Objekt hinzufügen. Zum Beispiel so:

```
var marker2 = L.marker([51.27264, 6.26469],  
{  
    title:"Marker2",  
    alt:"Ich bin ein anderer Marker"  
}  
  
).bindPopup('Marker2');  
myLayerGroup.addLayer(marker2);
```

Entfernen können sie das `LayerGroup` Objekt – und somit alle zu ihr gehörenden Objekt auf einen Schlag – mit der Methode `removeLayer()`. In unserm Beispiel konkret mit der Zeile
`myLayerGroup.removeLayer(polyline);`.

Featuregruppen

Die Klasse `FeatureGroup` erweitert die Klasse `LayerGroup`. Während die Klasse `LayerGroup` eher Methoden zum Gruppieren von Layern bereit stellt, geht es in der `FeatureGroup` hauptsächlich um das gemeinsame Verarbeiten von Ereignissen und das Hinzufügen von Stylen.

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Eine OSM Karte mit Leaflet</title>  
<link rel="stylesheet" href="../leaflet/leaflet.css" />  
<script src="../leaflet/leaflet.js"></script>
```

```

</head>

<body>

<div style="height: 700px;" id="mapid"></div>

<script>

var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var marker1 = L.marker([50.27264, 7.26469],
{
title:"Marker1",
alt:"Ich bin ein Marker"
}
);

var polyline = L.polyline([
[50.27264, 7.26469],
[51.27264, 7.26469],
[51.27264, 6.26469]],
);

var myfeatureGroup=L.featureGroup([marker1,
polyline]).addTo(mymap);

myfeatureGroup.bindPopup('Wir haben alle das gleiche Popup!');

</script>
</body>
</html>

<!--index_979.html-->

```

Analog zum Beispiel im Kapitel Layergruppen sehen Sie im vorhergehenden Programmcodeausschnitt, wie zwei Elemente erstellt und einem Objekt vom Typ FeatureGroup zugeordnet werden. Sie werden zusammen zum Karten Objekt hinzugefügt und ein Aufruf der Methode `bindPopup()` beim FeatureGroup Objekt fügt das Pop-up zu allen Objekten des FeatureGroup Objektes hinzu.

Wenn Sie alle Elemente des FeatureGroup Objektes rot färben möchten, reicht die Zeile `myfeatureGroup.setStyle({color:'red'})` aus.

Der Eintrag

```
myfeatureGroup.on('click', function()
{
  alert('Ein Gruppenmitglied wurde angeklickt!');
})
```

würde bewirken, dass sich immer dann, wenn ein Element des FeatureGroup Objektes angeklickt wird, ein Hinweisfenster öffnet.

Wenn Sie das FeatureGroup Objekt und dessen Inhalt mit einem Mausklick los werden möchten, können Sie folgenden Text zu Ihrem Skript hinzufügen:

```
myfeatureGroup.on('click', function()
{
  myfeatureGroup.removeLayer(polyline);
})
```

Popups

In diesem Kapitel geht es um Leaflet Objekte vom Typ [Popup](#). Mithilfe dieses Objekts können Sie Pop-up Fenster an bestimmten Stellen auf Ihrem Karten Objekt öffnen. Wenn Sie die Methode [openPopup\(\)](#) einsetzen, um das Pop-up zu öffnen, stellt Leaflet sicher, dass gleichzeitig nur ein Pop-up geöffnet ist. Ich empfehle Ihnen diese Methode einzusetzen, weil ich der Meinung bin, das das gleichzeitige Öffnen von mehreren Pop-up Fenstern nicht benutzerfreundlich ist. Nichtsdestotrotz kann es aber sein, dass Sie gerne mehrere Pop-up Fenster gleichzeitig anzeigen möchten. In diesem Fall müssen Sie das Pop-up mit der Methode [addLayer\(\)](#) auf einem eigenen Layer einbinden.

So nun aber genug der Theorie. Sehen Sie sich das alles anhand des nachfolgenden Programmcodes – am besten auf praktische Weise – an.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick:
false}).setView([50.27264, 7.26469], 7);
```

```

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var marker1 = L.marker([50.27264, 7.26469],
{
title:"Marker1",
alt:"Ich bin Marker 1"
}
).addTo(mymap);

var marker2 = L.marker([51.27264, 6.26469],
{
title:"Marker2",
alt:"Ich bin Marker 2"
}
).addTo(mymap);

marker1.bindPopup("<h1>Gering</h1><p>Ein kleines <a
href='https://de.wikipedia.org/wiki/Dorf'>Dorf</a></p><ul><li>auf
dem Maifeld</li><li>in der Eifel</li><li>an der Elz</li></ul>");
marker2.bindPopup("<h1>Boisheim</h1><p>Ein kleines
Dorf</p><ul><li>irgendwo</li><li>nordwestlich</li><li>von
Gering</li></ul>");

</script>
</body>
</html>
<!--index_978.html-->

```

Was zeigt Ihnen dieses Beispiel genau? Sie sehen, wie Sie zwei Marker auf erstellen und diesen mit der Methode `bindPopup()` ein Pop up Fenster hinzufügen könne. Wenn Sie den ersten Marker anklicken, dann öffnet sich das Pop-up Fenster dieses Markers. Wenn Sie danach den zweiten Marker anklicken, schließt sich das Pop-up des ersten Markers und das Pop-up des zweiten Markers wird aktiv. Vielleicht ist Ihnen aufgefallen, dass wir beim Karten Objekt die Option `closePopupOnClick` auf `false` gesetzt haben. Diese Option ist standardmäßig mit `true` belegt, was bedeutet, dass auch beim Klick auf eine beliebige Position auf der Karte ein eventuell offenes Pop-up von Leaflet automatisch geschlossen wird.

Möchten Sie, dass das Pop-up zum Marker mit dem Namen `marker1` schon beim Öffnen der Karte angezeigt wird? Dann schreiben Sie einfach den Text `marker1.openPopup();` in Ihr Skript.

Wie die Karte zum vorhergehend Programmcodeabschnitt im Browser aussieht, sehen Sie in der nachfolgenden Abbildung.

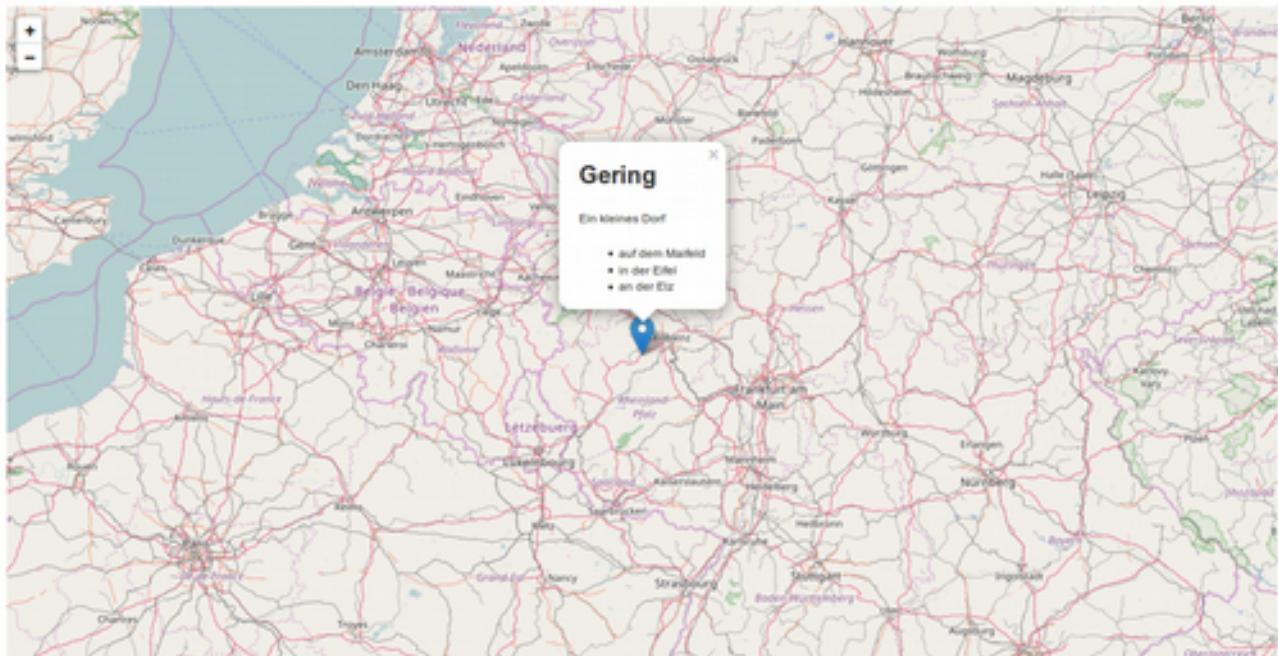


Abbildung 13

985.png

Hinweis:

Ich hatte Ihnen ja zu Beginn dieses Kapitels schon erklärt, dass Leaflet standardmäßig nur ein Pop-up gleichzeitig offen anzeigt. Dies ist so, wenn Sie das Pop-up mithilfe der Methode `openOn()` zu Karte hinzufügen:

```
var popup1 = L.popup({keepInView:true})
.setLatLng([49.27264, 6.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein Popup.</p>')
.openOn(mymap);
```

Falls Sie mehrere Pop-up Fenster gleichzeitig anbieten möchte, dann können Sie dies mithilfe der Methode `addTo()` bewerkstelligen.

```
var popup1 = L.popup()
.setLatLng([49.27264, 6.26469])
.setContent('<p>Hallo Welt<br />Ich bin ein Popup.</p>')
.addTo(mymap);
```

Die Eigenschaft `keepInView:true` bewirkt hier übrigens, dass die Karte nur so weit verschoben werden kann, wie der Marker sichtbar ist.

Mobil

Ein großer Vorteil von JavaScript ist es, das Landkarten in jedem aktuellen Standardbrowser ohne notwendige externe Applikationen oder Plugins angezeigt werden können.

Genaue Informationen zu den unterstützten Browsern finden Sie auf der Website von [Leaflet](#) im Bereich *Features*.

Jede Website, die eine Leaflet Karte anzeigt, kann vom Entwickler auf die gleiche Art und Weise programmiert werden. Der Entwickler muss im Normalfall nichts Spezielles für ein Gerät beachtet. Ein paar erwähnenswerte Punkte gibt es aber trotzdem und das was meiner Meinung nach erwähnenswert ist finden Sie in diesem Kapitel.

Das Spannendste bei der Arbeit mit mobilen Anwendungen ist sicherlich die Funktion [`locate\(\)`](#). Diese Funktion versucht, den Benutzer mit der W3C Geolocation API zu lokalisieren. Die [W3C Geolocation API](#) ist eine einheitliche Webbrowser-Programmierschnittstelle zum Ermitteln des geografischen Standorts des zugehörigen Endgeräts. Aber beginnen wir von vorne mit dem Bereich HTML und CSS.

HTML und CSS

Sie möchten Ihre Leaflet Karte auch für mobile Geräte optimal konfigurieren? Dann sollten Sie als Erstes sicherstellen, dass die Karte passend angezeigt wird. Sie möchten sicher nicht, dass jemand der die Karte über ein Gerät mit einem kleinen Display aufruft, nur einen Teil der Karte sieht und er diese erst verschieben muss, um auch die Randbereiche zu erkennen. Oder, dass die Karte so klein dargestellt wird, dass der Inhalt auf einem kleinen Display nicht lesbar ist. Ideal ist es, wenn die vollständige Karte im Display lesbar angezeigt wird. In diesem Punkt stimmen Sie mir sicher zu! Das nächste Beispiel zeigt Ihnen, wie Sie diesen Punkt umsetzen können.

```
<!DOCTYPE HTML>
<html>
```

```
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, user-scalable=no" />
<style>
body {
padding: 0;
margin: 0;
}
html, body, #mapid {
height: 100vh;
width: 100vw;
}
</style>
</head>
<body>
<div id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick:
false}).setView([50.27264, 7.26469], 18);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
mymap.locate({setView: true, maxZoom: 18});
mymap.on('locationfound', onLocationFound);
mymap.on('locationerror', onLocationError);

function onLocationFound(e) {
var radius = e.accuracy / 2;
L.marker(e.latlng).addTo(mymap).bindPopup("Sie sind etwa" + radius
+ " Meter von diesem Punkt entfernt.").openPopup();
L.circle(e.latlng, radius).addTo(mymap);
}
function onLocationError(e) {
alert(e.message);
}
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

```
<!--index_977.html-->
```

Hinweis:

Wissen Sie, dass Sie zum Debuggen die Methode `console.log()` verwenden können?

Der Eintrag

```
<script>
..
console.log(e);
..
</script>
```

gibt an einer Stelle, an das Objekt `e` instanziert ist, den Inhalt dieses Objektes aus.

Die Konsole können Sie im Browser [Mozilla Firefox](#) über das Menü Extras | Web Entwickler | Web Konsole öffnen. Im Browser [Google Chrome](#) finden Sie die Konsole über das Menü Tools | Javascript-Konsole.

Dieses Beispiel enthält zusätzlich zu den vorhergehenden Beispielen ein `<style>`-Element. In diesem sind die Abstände, also `padding` und `margin`, auf 0 gesetzt. So passt die Karte sich genau an die Seite an und es wird kein Platz mit Rändern verschwendet. Außerdem haben wir die Höhe auf 100 % gesetzt, damit diese voll ausgenutzt wird.

Hinweis:

Die Einheiten [vw](#) und [vh](#) definieren eine Breite beziehungsweise Höhe in Relation zur Größe des Browser-Fensters. Dabei steht `vw` für die Breite und `vh` für die Höhe. Diese sogenannten Viewport Units ermöglichen es, Größen in Relation zur jeweils aktuellen Größe des Browser-Fensters zu definieren.

Den fixen Wert für die Höhe, der im `<div>`-Element des Karten-Objektes enthalten war, habe ich entfernt. So verschwindet auch die Bildlaufleiste am rechten Rand.

Außerdem ist in diesem Beispiel das Meta-Element `viewport` hinzugekommen. Somit kann die Seite nicht verkleinert oder vergrößert werden – zoomen ist aber immer noch möglich!

Hinweis:

Sehr wichtig im Bereich der mobilen Entwicklung ist Meta-Element `viewport`. In unserm Beispiel haben wir die Attribute `initial-scale=1.0`, `maximum-scale=1.0`, `user-scalable=no` verwendet.

Das Attribut `initial-scale` legt die anfänglichen Zoom-Stufe fest. `1.0` führt dazu, dass die Inhalte in ihrer Originalgröße dargestellt werden. Das bedeutet, dass auf einem Display mit 320 Pixel Breite, eine 320 Pixel breite Grafik die komplette Breite ausfüllen würde. Im Gegensatz dazu würde der Wert `2.0` zu einer 2-fachen Vergrößerung führen – das Bild wäre nur noch halb zu sehen.

Das Attribut `user-scalable` bestimmt, ob der Nutzer die Anzeige der Website vergrößern oder verkleinern kann.

Die Attribute `minimum-scale` und `maximum-scale` ermöglichen es die Zoom-Stufe einzuschränken. Die Vorgabe `maximum-scale =1.0`, bewirkt, dass der Inhalt nicht vergrößert werden kann.

JavaScript

Die Karte wird nun passend auf der Website angezeigt. Soweit so gut! Wir können die Karte aber noch benutzerfreundlicher machen. Jemand der mit einem mobilen Gerät im Internet unterwegs ist, nutzt sein Gerät in der Regel an verschiedenen Standorten. Und meistens ist es so, dass das was in der Nähe ist, am interessantesten ist. Ideal wäre es also, wenn die Karte sich sofort so öffnet, dass der aktuelle Standort in der Mitte der Karte dargestellt wird. Und das ist einfach möglich. Leaflet bietet Ihnen Funktionen, die Sie nutzen können.

```
<!DOCTYPE HTML>

<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, user-scalable=no" />
<style>
body {
padding: 0;
margin: 0;
}
```

```

html, body, #mapid {
height: 100vh;
width: 100vw;
}

</style>
</head>
<body>
<div id="mapid"></div>
<script>

var mymap = L.map('mapid', {closePopupOnClick:
false}).setView([50.27264, 7.26469], 18);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

mymap.locate({setView: true, maxZoom: 18});

mymap.on('locationfound', onLocationFound);
mymap.on('locationerror', onLocationError);

function onLocationFound(e) {
var radius = e.accuracy / 2;
L.marker(e.latlng).addTo(mymap).bindPopup("Sie sind etwa" + radius
+ " Meter von diesem Punkt entfernt.").openPopup();
L.circle(e.latlng, radius).addTo(mymap);
}

function onLocationError(e) {
alert(e.message);
}

</script>
</body>
</html>
<!--index_977.html-->

```

Wenn Sie eine Website besuchen, die standortbezogenes Surfen unterstützt – zum Beispiel die Leaflet Karte im vorherigen Beispielcode – prüft der Browser, ob Sie Ihren Aufenthaltsort bekannt geben möchten. Wenn Sie zustimmen, berechnet der Browser Ihren Standort über nahe gelegene Funkzugangsknoten und die IP-Adresse Ihres Computers. Diese ungefähre Ortsangabe wird dann an die anfragende Website weitergegeben.

Wenn Sie Ihren Standort freigegeben haben, dann sehen beim Aufruf der Karte einen Kreis, der Ihren ungefähren Standort wiedergibt und ein Pop-up, dass Ihnen erklärt wo Sie sich gerade befinden. Möchten Sie Ihren Standort nicht mit anderen teilen, dann sehen Sie eine Fehlermeldung beim Öffnen der Karte, die Ihnen erklärt, warum Sie nicht geortet werden können.

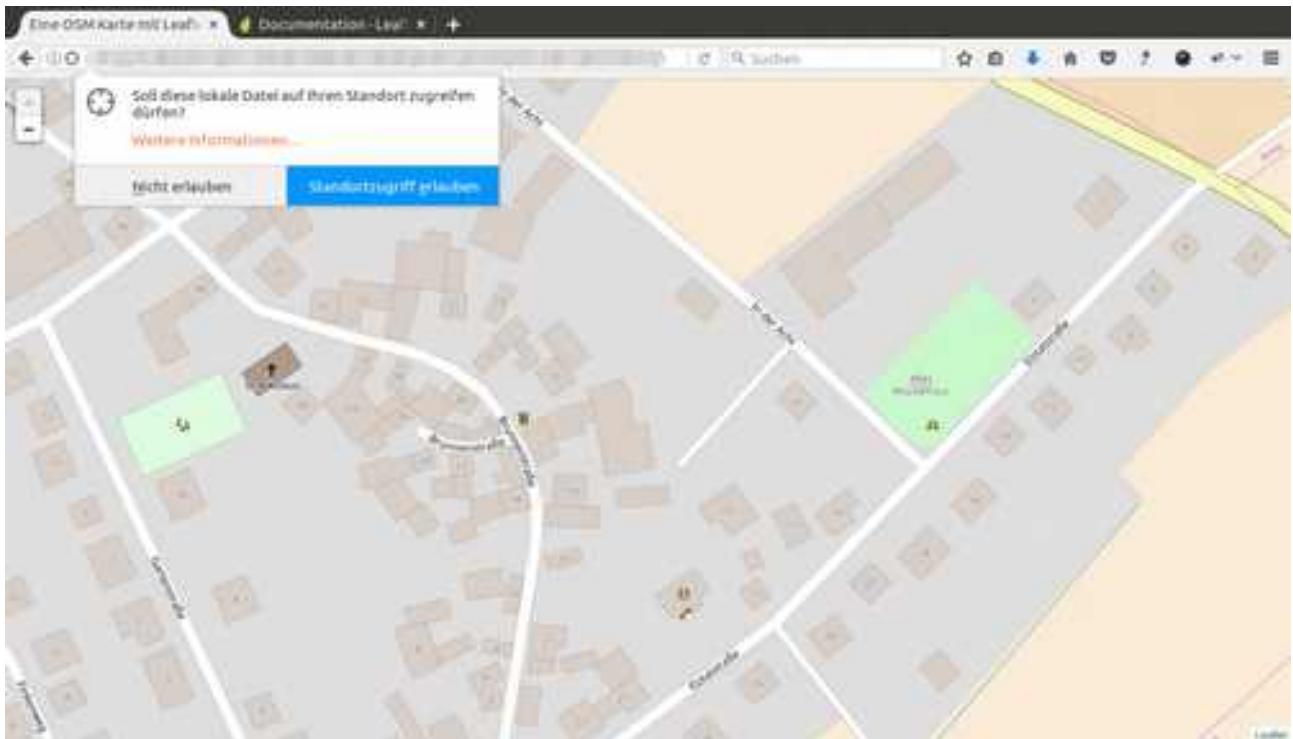


Abbildung 14

984.png

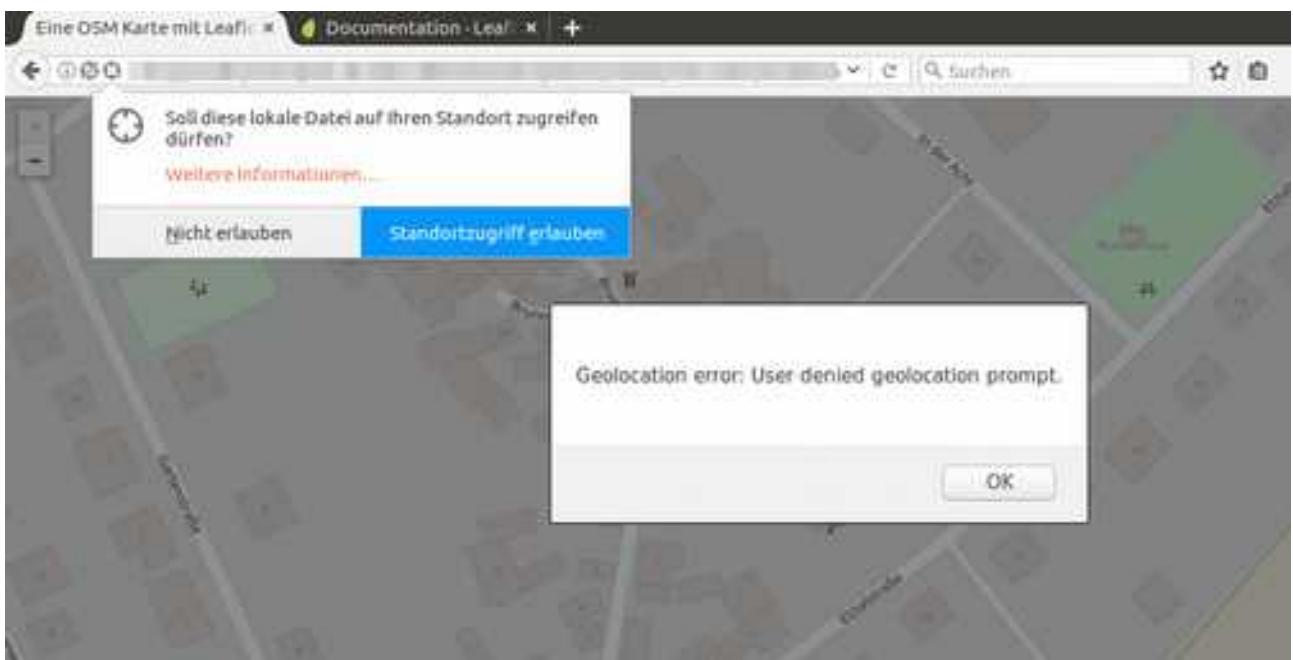


Abbildung 15

984a.png

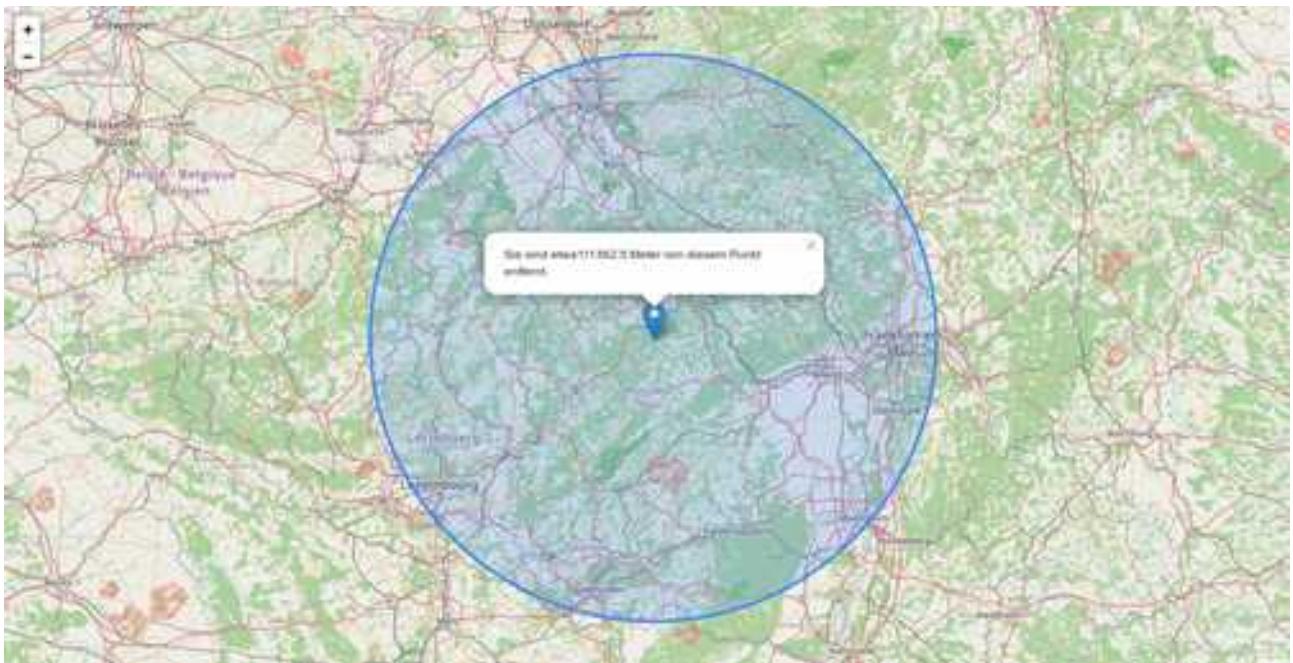


Abbildung 16
984b.png

Leaflet bietet Ihnen noch weitere [Geolokalisierungsmethoden](#) und [Geolokalisierungereignisse](#) an.

Ereignisse in Leaflet

Bisher haben wir ausschließlich mit statischen Daten gearbeitet. In der Welt passiert aber fortwährend ganz schön viel und Sie möchten sicherlich mit Ihrer Karte auf das ein oder andere Ereignis reagieren. Vielleicht möchten Sie je nach Standort einen Pop-up Text ändern – oder dann, wenn der Marker zum Pop-up bewegt wird. Oder Sie möchten auf ein anderes Ereignis reagieren. Vielleicht wissen Sie auch noch gar nicht so genau auf welche Ereignisse Sie reagieren können. Dann schen Sie sich zur Inspiration doch einfach einmal an, was Leaflet Ihnen bietet. Insgesamt bietet Leaflet Ihnen 34 verschiedene Ereignisse, die Sie nutzen können. Alle Ereignisse sind der [Dokumentation](#) gut erklärt.

Exemplarisch sehen wir uns mit dem folgenden Programmcode – wieder anhand eines ganz einfachen Beispiels – die Vorgehensweise genauer an. Konkret reagieren wir auf einen Mausklick.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
```

```

<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick:
false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

mymap.on('click', function(){alert('Sie haben auf die Karte
geklickt.'}});
</script>
</body>
</html>
<!--index_976.html-->
```

Leaflet bietet die Methode [on\(\)](#) um auf ein Ereignis zu reagieren. Der erste Parameter, der mit dieser Methode übergeben wird, steht für den Ereignistyp. In unserem Falle ist die `click`. Der zweite Wert erwartet eine Funktion, die beim Eintreten des Ereignisses ausgeführt werden soll.

Hinweis:

In unserem Beispiel haben wir die Funktion sofort in den Methodenaufruf eingefügt:

```
mymap.on('click', function(){alert('Sie haben auf die Karte
geklickt.')});
```

Diese Schreibweise wird auch als anonymer Funktionsausdruck bezeichnet, weil die Funktion keinen Namen hat.

Wenn Sie die Karte des vorhergehenden Beispiels im Browser aufrufen und anklicken, sehen Sie die in der nachfolgenden Abbildung dargestellte Meldung.

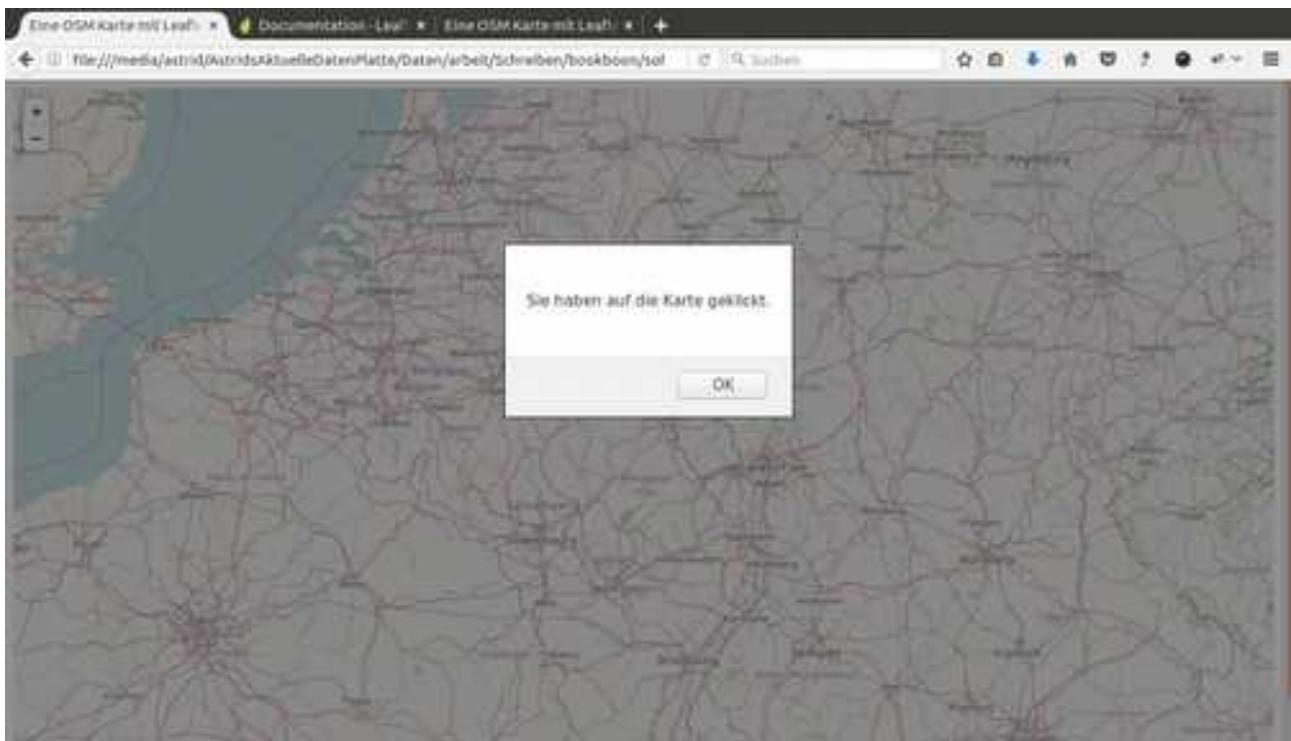


Abbildung 17

983.png

Hinweis:

Vielleicht möchten sie mittlerweile selbst eigene Beispiele ausprobieren und diesen vielleicht auch mit anderen teilen. Insbesondere bei der Fehlersuche ist dies oft hilfreich. Sie können die ganz einfach mithilfe des [Leaflet Playgrounds](#) tun.

Eine benutzerdefinierte Funktionen

Die Methoden, die Leaflet Ihnen bietet, kennen Sie nun. Bisher haben Sie immer die Variable `e` in den Funktionen übergeben. Mit JavaScript können Sie aber auch eigene Variablen übergeben. Außerdem können Sie eigene Funktionen an beliebigen Stellen im Programmcode aufrufen.

Probieren wir das doch sofort einmal aus. Im nächsten Beispiel erstellen wir einen Marker und verbinden diesen mit einem Pop-up – und das mithilfe einer selbst erstellten Funktion.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
```

```

<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid', {closePopupOnClick: false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var marker1 = L.marker([50.27264,
7.26469]).addTo(mymap).bindPopup(createPopup("Text als
Parameter"));
var marker2 = L.marker([51.27264,
6.26469]).addTo(mymap).bindPopup(createPopup("Anderer Text als
Parameter"));
function createPopup(popuptext) {
return L.popup({autoClose:false, keepInView:true,
closeButton:false}).setContent(popuptext);
}
</script>
</body>
</html>
<!--index_975.html-->

```

975.html

In diesem Beispiel ging es nur um das Grundsätzliche und die Mächtigkeit von eigenen Funktionen wird noch nicht ganz klar. Vielleicht können Sie sich aber trotzdem schon vorstellen, wie viel Text mithilfe einer benutzerdefinierten Funktion eingespart werden kann.

Achtung:

Wenn Sie bei einem Popup Objekt die Eigenschaften `autoClose:false` und `closeButton:false` gleichzeitig setzen, kann dieses Pop-up Fenster tatsächlich nicht mehr geschlossen werden.

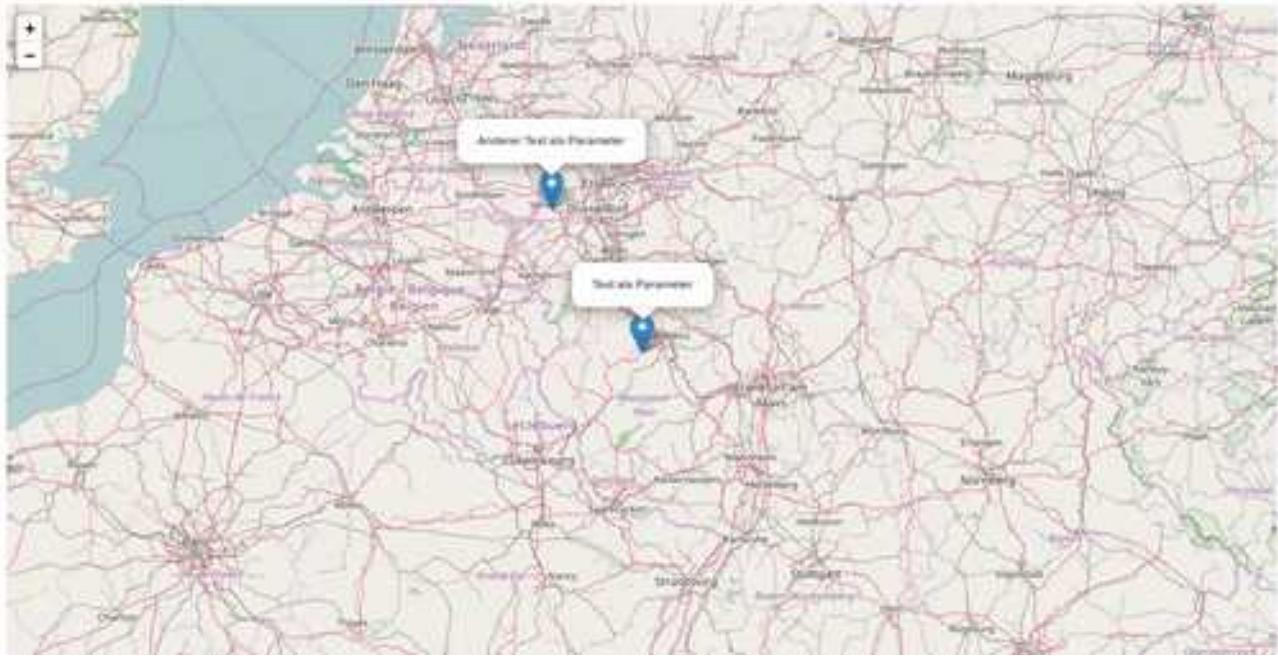


Abbildung 18
982.png

Ein interaktives Beispiel

Und zu guter Letzt möchte ich Ihnen noch ein interaktives Beispiel zeigen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<p>Bitte geben Sie eine Koordinate an:</p><br>
Geographische Breite:<input type="text" id="lat"><br>
Geographische Länge:<input type="text" id="long"><br>
Name:<input type="text" id="name"><br>
<b><button onclick="save()">Speichern</button></b>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
```

```

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

function save()
{
var p1 = parseFloat(document.getElementById("lat").value);
var p2 = parseFloat(document.getElementById("long").value);
L.marker([p1,p2]).bindPopup(document.getElementById("name").value)
.addTo(mymap);
}
</script>
</body>
</html>
<!--index_974.html-->
```

Im vorhergehenden Beispiel ist eine Schaltfläche integriert, die – wenn sie angeklickt wird – eine benutzerdefiniert Funktion ausführt. Mit der Funktion wird ein Marker mit einem Pop-up in die Karte eingesetzt. Dies ist nur als Beispiel gedacht. Die können in der benutzerdefinierten Funktion alles Mögliche ausführen – lassen Sie Ihrer Fantasie freien Lauf. Ein weiterer Anwendungsfall könnte eine Art Heiß-Kalt Spiel sein. Sie kennen sicher das einfache Suchspiel für Kinder.

Hinweis:

Möchten die interaktiv eingefügten Daten in einem Objekt speichern, um einen Überblick über diese zu haben und eventuell weitere Aktionen von dem Datenbestand abhängig zu machen?

Das vorhergehende Beispiel könnten Sie in diesem Falle wie folgt erweitern.

Sie könnten mithilfe der Zeile

```
var myStorage = localStorage;
ein localStorage Objekt instanziieren. Mit
myStorage.setItem(document.getElementById("name").value,
document.getElementById("lat").value+", "+document.getElementById("long").value);
```

könnten Sie daraufhin alle eingefügten Objekte speichern und mit diesen arbeiten.

Die `sessionStorage`-Eigenschaft erlaubt den Zugriff auf ein, nur während der aktuellen Sitzung, verfügbares Storage-Objekt. `sessionStorage` ist mit einer Ausnahme identisch zu `Window.localStorage`: In `localStorage` gespeicherte Daten besitzen kein Verfallsdatum, während Sie im

`sessionStorage` mit Ablauf der Sitzung gelöscht werden. Eine Sitzung endet erst mit dem Schließen des Browsers, sie übersteht das Neuladen und Wiederherstellen einer Webseite. Das Öffnen einer Webseite in einem neuen Tabulator oder Browser-Fenster erzeugt jedoch eine neue Sitzung. Dies unterscheidet ein `sessionStorage` Objekt von einem Session-Cookie.
<https://developer.mozilla.org/de/docs/Web/API/Window/sessionStorage>

In diesem Kapitel haben wir ...

In diesem Kapitel haben Sie Punkte, Marker, Linien und Polygone kennengelernt. Sie können diese nun auf eine Leaflet-Karte zeichnen und wissen, wann welches Objekt das Richtige ist. Sie können Layer-Gruppen und Feature-Gruppen voneinander unterscheiden und wissen die Grundlagen zur Anzeige von mobilen Leaflet Karte. Und auf einen Mausklick oder ein anderes Ereignis können Sie entsprechend reagieren. Im nächsten Kapitel geht es um das Format GeoJson und darum, wie Sie Daten auch in großen Mengen gut Handhaben können.

GeoJSON

GeoJSON ist ein offenes Format, welches es einfach macht, geografische Daten zu beschreiben. Dabei richtet es sich nach Spezifikation – nämlich nach der Simple-Feature-Access-Spezifikation. Für die Beschreibung der Geodaten verwendet GeoJSON die JavaScript Objekt Notation (JSON).

Hinter dem Begriff [Simple Feature Access-Spezifikation](#) versteckt sich eine Spezifikation des [Open Geospatial Consortium](#) (OGC). Diese Spezifikation beinhaltet eine allgemein gültige Beschreibung für Geodaten und deren Geometrien. Dadurch, dass die Spezifikation allgemein gültig ist, können diese Daten ausgetauscht werden. Das OGC ist eine gemeinnützige Organisation, die die Entwicklung von allgemeingültigen Standards für Geodaten zum Ziel hat.

In diesem Kapitel werden wir ...

Als Erstes sehen wir uns an, warum GeoJSON entwickelt wurde. Als Nächstes vergleichen wir die einzelnen Elemente mit den Objekten, die uns Leaflet zur Verfügung stellt. Und zu guter Letzt probieren wir die Methoden aus, die Leaflet Ihnen – spezielle für das Verarbeiten von GeoJSON-Daten – anbietet.

Die Entwicklungsgeschichte von GeoJSON

GeoJSON baut auf [JSON](#) auf und bevor JSON als Datenformat spezifiziert wurde, gab es die erweiterbare Auszeichnungssprache [XML](#) (englisch Extensible Markup Language). Immer wenn etwas Neues entsteht, gibt es einen Grund dafür. XML wurde 1998 [veröffentlicht](#), um Daten zwischen Maschinen austauschen zu können, ohne das Menschen nacharbeiten müssen. Dies wurde in Zeiten des Internets immer wichtiger. Nun muss es auch einen Grund geben, warum neben XML JSON – und später GeoJSON entstanden ist.

Warum ist das Format GeoJSON entstanden?

Warum hat XML für den Austausch von Daten nicht ausgereicht und welche Vorteile bietet JSON, beziehungsweise GeoJSON? Zunächst einmal bieten alle drei Formate folgenden:

- Alle drei Formate können von einem Menschen gelesen und verstanden werden.
- Alle drei Formate sind hierarchisch gegliedert. Das bedeutet, dass Werte innerhalb von anderen Werten dargestellt werden können.
- Alle drei Formate sind relativ leicht zu erlernen.
- Alle drei Formate können von vielen Programmiersprachen analysiert und genutzt werden.
- Alle drei Formate sind mit mithilfe des Hypertext Transfer Protokolls (HTTP) – also über das Internet – austauschbar.

Wenn wir uns in den nächsten Kapiteln die einzelnen Formate genauer ansehen wird Ihnen klar werden, welche Vorteile das Format JSON – beim Arbeiten mit [Geodaten](#) das Format GeoJSON – gegenüber XML bringt.

XML

XML beschreibt die Struktur von Daten. Anhand der Tags wird den Daten eine Bedeutung – eine Semantik – gegeben. Durch das Tag-System von XML werden oft kleine Datenbestände sehr aufgebläht und unübersichtlich. Zusätzlich ist das Ansprechen einzelner Tags in einer XML-Datei nicht immer leicht.

JSON

JSON ist im Grunde genommen nichts anderes als die Festlegung auf eine bestimmte Syntax – also eine Syntax-Konvention. Den Daten wird keine bestimmte Bedeutung geben, vielmehr geht es um die syntaktische

Anordnung. Da JSON Daten strukturiert können leicht Objekte aus diesen Daten definiert werden. Im Dezember 1999 wurde die erste [JSON Format-Spezifikation](#) verabschiedet. Im Juni 2017 wurde die 8. Edition des Standards ECMA-262 veröffentlicht.

Der große Vorteil von JSON im Vergleich zu XML liegt in der einfachen Handhabung. Da JSON selbst gültiges Javascript darstellt, kann es direkt ausgeführt und somit in ein Javascript-Objekt überführt werden. Auf die einzelnen Eigenschaften dieser Objekte kann dann über Attribut zugegriffen werden. Im Kapitel *Heatmaps in Leaflet – Dichte* werden wir eine Datei, die GeoJSON Objekte enthält, als Skript einbinden und können nur das Einbinden innerhalb anderer Skripts auf die GeoJSON Objekte der eingebundenen Datei zugreifen. Im Gegensatz dazu muss eine XML-Datei erst mit einem XML-Parser analysiert werden.

Ein weiterer Vorteil von JSON: In JSON gibt es kein Ende-Tag, hauptsächlich deshalb ist JSON kompakter. Dies hat zur Folge, dass JSON schneller gelesen und verarbeitet werden kann.

Hinweis:

Die Daten einer jeden GeoJson-Datei, die sich in einem GitHub Repository befindet, werden – wenn man die Datei im Repository anklickt – automatisch auf einer interaktiven Karte angezeigt. [Github](#) erstellt diese Karte schon seit 2013 mit Leaflet. Dies können Sie sich beispielsweise im Repository [world.geo.json](#) ansehen.

Schon ein kleines Beispiel veranschaulicht, dass XML für das Beschreiben des gleichen Objektes mehr Zeichen benötigt als JSON. Ein in XML mit 95 Zeichen kodiertes Objekt benötigt in JSON gerade einmal 73 Zeichen. Bei einem Objekt ist dieser Unterschied vernachlässigbar. In der Regel werden aber eine Vielzahl von Objekten digital beschrieben und bei einer Vielzahl von Objekten kann dieser Unterschied stark ins Gewicht fallen.

Hier sehen Sie zunächst den aus 95 Zeichen bestehende XML Ausschnitt.

```
<joomlers>
<number>1721</number>
<vorname>Astrid</vorname>
<nachname>Günther</nachname>
</joomlers>
```

Das gleiche Objekt im JSON Format kann mit 73 Zeichen beschrieben werden.

```
„joomlers“: {  
    „number“: „1721“,  
    „vorname“: „Astrid“,  
    „nachname“: „Günther“  
},
```

Und warum nun auch noch GeoJSON?

Geodaten könnten in JSON beschrieben und verarbeitet werden. Welchen Vorteil bringt das spezielle GeoJSON-Format? GeoJSON ist JSON – allerdings auf Geodaten spezialisiert. GeoJSON gibt den GeoDaten wieder eine Semantik – also eine Bedeutung. GeoJSON beschreibt Punkten, Linien und Polygonen und gut mit diesen in einem Koordinatensystem umgehen. Im vorausgehenden Kapitel haben wir gesehen, dass das Arbeiten mit Geodaten im Grunde genommen nichts anderes ist. GeoJSON hat sich zu einem sehr beliebten Datenformat vieler Geoinformationssysteme entwickelt. In diesem Buch erwähne ich GeoJSON speziell, weil auch Leaflet sehr gut im Umgang mit Daten im GeoJSON Format ist. Hier sehen wir uns zunächst die GeoJSON Objekte einmal genauer an. Wenn Sie lieber sofort praktisch arbeiten möchten, dann Blättern Sie am besten ein Kapitel weiter. Im Kapitel *GeoJson in Leaflet* erfahren Sie, wie Sie GeoJSON-Elemente auf Ihrer Karte anzeigen und weiter bearbeiten können. Im Juni 2008 wurde die erste Format-Spezifikation verabschiedet. Im August 2016 wurde die [RFC \(Requests for Comments\) 7946](#) veröffentlicht.

GeoJSON erkunden

Sie wissen nun, dass Sie mit GeoJSON viele geografische Datenstrukturen in einer maschinenlesbaren Sprache kodieren können. Ein GeoJSON-Objekt kann dabei eine einfache Geometrie, zum Beispiel einen Punkt eine Linie oder ein Polygon, darstellen. Zusätzlich können Sie einer Geometrie aber auch Eigenschaften zuordnen. In diesem Fall erstellen Sie ein Objekt vom Typ Feature. Wenn Sie mehrere Feature-Objekte zu einer Gruppe zusammen fassen möchten, können Sie diese zu einer Sammlung von Features zusammen fassen. Hierfür gibt es den GEOJson Typ FeatureCollection. Das Verständnis dieser Konzepte bringt viele Vorteile. Es hilft Ihnen auch, die Arbeit mit Geodaten im Allgemeinen zu verstehen: Die Grundkonzepte, die in GeoJSON angewandt werden, sind schon seit vielen Jahren ein Teil von [Geoinformationssystemen](#).

Die formale Spezifikation des GeoJSON Formates finden Sie unter der Adresse <https://tools.ietf.org/html/rfc7946> im Internet.

Eine Geometrie

Eine Geometrie ist eine Form. Alle Formen in GeoJSON werden mit einer oder mehreren Koordinaten beschrieben. Eine Koordinate heißt in GeoJSON Position. GeoJSON unterstützt die Geometriarten Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon – und diese beinhalten Positionen.

Position

Das wichtigste Element beim Arbeiten mit Geodaten ist die Beschreibung des Punktes auf der Erde, dem die Geodaten zugeordnet werden. Diesen Wert kennen wir auch unter dem Namen Koordinate. Im Kapitel *Das Koordinatensystem der Erde* hatte ich schon eine ganze Menge zum Thema Koordinaten auf der Erde geschrieben. Hier noch einmal kurz: Eine Koordinate ist eine Zahlenkombination. Jede Zahl einer Koordinate steht für eine Dimension. Wir beschränken uns in diesem Buch auf zwei Dimensionen, nämlich die geografische Längen und die geografische Breite. GeoJSON unterstützt drei Dimensionen – neben der geografischen Länge und der geografischen Breite können Sie zusätzlich die Höhe auf der Erde angeben.

Beim [globales Navigationssatellitensystem](#) (GPS) ist noch eine vierte Große relevant. Neben der horizontalen Position und der Höhe spielt auch die aktuelle Zeit eine Rolle.

Die Koordinaten werden in GeoJSON im Dezimalformat formatiert. Die erste Zahl steht für die Longitude – also die geografische Länge – und die zweite Zahl für die Latitude – also die geografische Breite. Konkret sieht eine Position in GeoJSON so aus:

[Länge, Breite, Höhe] oder [50.254, 7.5847, 324.1]

Vielleicht haben Sie in der Vergangenheit schon öfter mit Geodaten gearbeitet und wundern sich nun über die Reihenfolge, in der die Dimensionen im Format GeoJSON stehen. Viele alt eingesetzte Systeme geben zuerst die geografische Länge und erst dann die geografische Breite an. Auch in Leaflet wird bei der Koordinate zuerst die Latitude, also die geografische Länge, und erste dann die Longitude, also die geografische Breite, angegeben. Um dieses Durcheinander zu verstehen, müssen Sie

folgendes bedenken: Früher war es üblich, dass die erste Stelle einer Koordinate den Breitengrad und die zweite Stelle den Längengrad beschrieb. In der Mathematik ist die übliche Reihenfolge beim Arbeiten mit Koordinatensystemen: X-Wert | Y-Wert. Wenn man eine Landkarte mit einem Koordinatensystem vergleicht, erkennt man schnell, dass der Breitengrad dem X-Wert und der Längengrad der dem Y-Wert entspricht. Dies hat zur Folge, dass es beim Rechnen mit einem Computer viele Vorteile bringt, wenn man die Reihenfolge Längengrad | Breitengrad einhält. In der digitalen Welt gibt es momentan noch keine Einigkeit über die Reihenfolge. Es sieht so aus, als wir da mitten in einem Umbruch stecken. Eine Übersicht, die zeigt, welche Anwendung welche Dimension als erstes verwendet, finden Sie unter anderem unter der Adresse <https://macwright.org/lonlat/>.

Früher erlaubte GeoJSON die Speicherung von mehr als 3 Zahlen pro Position. Diese Möglichkeit wurde auch genutzt. Es wurden beispielsweise Sportdaten wie die Herzfrequenz zusammen mit der Position gespeichert. Da dies nicht der Sinn einer Position ist, führte dieses Vorgehen teilweise zu Problemen in anderen GeoJSON Anwendungen. In der [neuen Spezifikation](#) ist das Speichern von mehr als drei Werten pro Position nun nicht mehr zulässig.

Point

Der Typ `Point` – also ein Punkt – ist die einfachste Geometrie in GeoJSON. Er gibt die Koordinaten einer bestimmten Position an. Die genaue Schreibweise sehen Sie nachfolgende.

```
{ "type": "Point",
  "coordinates": [30, 10]
}
```

In Leaflet wird der Typ `Point` als Marker eingefügt.

Multipoint

Der Typ `MultiPoint` wird mit einem Array von Positionen beschrieben. Mit ihm können mehrere Punkte auf der Erde angegeben werden.

```
{ "type": "MultiPoint",
  "coordinates": [
    [10, 40], [40, 30], [20, 20], [30, 10]
```

```
]  
}
```

Mit dem Typ `Multipoint` können Sie mehrere Marker auf einen Schlag zu Ihrer Leaflet Karte hinzufügen.

LineString

Um eine Linie darzustellen, benötigen Sie mindestens zwei Punkte. Die Linie ist die Verbindung zwischen diesen Punkten. Eine Linie wird mit einem Array von zwei oder mehr Positionen dargestellt. In GeoJSON wir eine Linie mit dem Typ `LineString` dargestellt.

```
{ "type": "LineString",  
  "coordinates": [  
    [30, 10], [10, 30], [40, 40]  
  ]  
}
```

Ein GeoJSON Objekt vom Typ `LineString` entspricht einem `Polyline` Objekt in Leaflet.

MultiLineString

Beim Typ `MultiLineString` werden die Koordinaten mit einem Array von `LineString`-Koordinaten-Arrays angegeben.

```
{ "type": "MultiLineString",  
  "coordinates": [  
    [[10, 10], [20, 20], [10, 40]],  
    [[40, 40], [30, 30], [40, 20], [30, 10]]  
  ]  
}
```

Ein GeoJSON Objekt vom Typ `MultiLineString` entspricht einem Leaflet `Polyline` Objekt, welches mehr als eine abgeschlossene Linie definiert. Das bedeutet, dass alle Linien zusammen auf einen Layer gezeichnet werden.

Wie bei den Objekten in Leaflet gilt auch hier: Linien und Punkte sind die einfachsten Geometrieverbände. Bei beiden müssen Sie nicht viele geometrische Regeln beachten. Ein Punkt kann irgendwo an einem Ort liegen und eine Linie kann eine beliebige Anzahl an Punkten enthalten. Eine Linie kann sich selbst überqueren. Punkte und Linien haben keine Fläche – somit gibt es auch kein Außen und kein Innen.

Polygone

Im Vergleich zu Linien sind Polygone komplexe Geometrien. Polygone verfügen über eine Fläche. Es gibt also einen Innenbereich, der sich von einem Außenbereich unterscheidet. Und hinzu kommt: Der Innenbereich kann Löcher haben! Wie die Löcher in einem Polygon entstehen habe ich Ihnen im Kapitel *Die Karte mit Daten bestücken* im Unterkapitel *Polygone* bereits erklärt. Ein GeoJSON Objekt vom Typ `Polygon` entspricht einem `Polygon` Objekt in Leaflet. Die Koordinatenliste enthält bei einem `Polygon` – analog zu Leaflet – eine Ebene mehr als der Typ `LineString`. Ich habe die relevante Klammerung im nachfolgenden Beispiel fett formatiert.

```
{ "type": "Polygon",
  "coordinates": [
    [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]]
  ]
}
```

Bei einem einfachen Polygon ist der Sinn dieser Ebene nicht offensichtlich. Auf den ersten Blick könnten Sie der Meinung sein, dass es einfacher wäre, das Polygon genau wie die Linie zu erstellen. Dass es sich um ein Polygon handelt, ist über den Eintrag bei der Eigenschaft Typ klar – und wenn es sich um den Typ `Polygon` handelt, wird die Linie einfach geschlossen! Der erste Blick ist oft trügerisch. Wir benötigen diese zusätzliche Möglichkeit der Klammerung oder Verschachtelung um Löcher in die Fläche zeichnen zu können. Polygone sind in GeoJSON mehr als nur geschlossene Linien. Ich wiederhole mich. Polygone haben einen Innenbereich und dieser kann Löcher haben. Aus diesem Grund ist

beim Typ `Polygone` in der GeoJSON Spezifikation ein neuer Begriff zu lesen, nämlich der Begriff `LinearRing`. Ein `LinearRing` ist ein geschlossener `LineString` mit vier oder mehr Positionen. Obwohl ein `LinearRing` nicht explizit als GeoJSON-Geometrie-Typ eingeführt ist, wird der Begriff in der [Polygon-Geometrie-Typ-Definition](#) erwähnt.

Ein `LinearRing` ist entweder die äußere Ringposition, die die äußere Kante des Polygons bildet und definieren, welche Teile gefüllt sind. Ein `LinearRing` kann aber auch ein Innenring sein, der die Teile des Polygons definieren, die leer sind. Es kann eine beliebige Anzahl von Innenringen geben, einschließlich null Innenringen. Wenn das Polygon über keinen Innenring verfügt bedeutet dies, dass das Polygon nur einen Innenbereich und einen Außenbereich hat – also keine Löcher hat.

```
{ "type": "Polygon",
  "coordinates": [
    [[1, 1], [1, 10], [10, 10], [10, 1], [1, 1]],
    [[2, 2], [2, 5], [5, 5], [5, 2], [2, 2]]
  ]
}
```

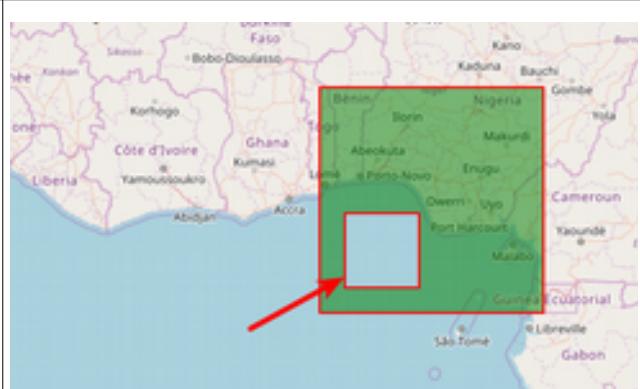


Abbildung 19: Ein Polygon mit einem Innenring. Der Innenring definiert einen Außenbereich im Polygon – er schneidet quasi ein Loch in das Polygon.

```
{
  "type": "Polygon",
  "coordinates": [
    [[1, 1], [1, 10], [10, 10], [10, 1], [1, 1]],
    [[2, 2], [2, 5], [5, 5], [5, 2], [2, 2]],
    [[3, 3], [3, 4], [4, 4], [4, 3], [3, 3]]
  ]
}
```



Abbildung 20: Ein Polygon mit zwei Innenringen – der zweite Innenring wird innerhalb des ersten Innenringes gezeichnet. Dieser zweite Innenring zeichnet einen neuen Innenbereich in den Außenbereich der durch den ersten Innenring entstanden ist.

Vielleicht ist Ihnen aufgefallen, dass die erste und die letzte Koordinate jedes LinearRings gleich ist. Die Wiederholung der Koordinate ist bei einem Leaflet Objekt nicht erwünscht. Hier werden die Ringe eines Polygone automatisch geschlossen. Die erste und letzte Position eines GeoJSON LinearRing muss im Gegensatz dazu identisch sein.

MultiPolygon

Beim Typ `MultiPolygon` werden die Koordinaten mit einem Array von Polygon-Koordinaten-Arrays angegeben. Hier sehen Sie zunächst ein Beispiel, dass zwei einfache Polygone darstellt.

```
{
  "type": "MultiPolygon",
  "coordinates": [
    [
      [[30, 20], [45, 40], [10, 40], [30, 20], [30, 20]]
    ],
    [
      [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]]
    ]
  ]
}
```

Es geht aber auch komplizierter. Sie können auch mehr als ein Polygon mit Löchern – also mehr als einem `LinearString` – zusammen als `MultiPolygon` gruppieren.

```
{ "type": "MultiPolygon",
  "coordinates": [
    [
      [[40, 40], [20, 45], [45, 30], [40, 40]]
    ],
    [
      [[20, 35], [10, 30], [10, 10], [30, 5], [45, 20], [20, 35]],
      [[30, 20], [20, 15], [20, 25], [30, 20]]
    ]
  ]
}
```

Ein GeoJSON Objekt vom Typ `MultiPolygon` entspricht einem Leaflet `Polygon` Objekt, welches mehr als ein Polygon definiert. Das bedeutet, dass alle Formen zusammen auf einen Layer gezeichnet werden.

Mehrere Geometrien zusammenfassen - `GeometryCollection`

Geodaten wollen die Welt beschreiben. Jeder der grundlegenden GeoJSON Typen ist ideal für die Darstellung eines Objektes auf der Erde. Unsere Welt enthält eine Menge Objekte, die gemeinsame Eigenschaften haben. Wenn wir diesen Objekten diese gemeinsamen Eigenschaften auf einen Schlag zuordnen möchten, können wir diese Objekte mit dem Typ `GeometryCollection` zusammenfassen.

```
{
  "type": "Feature",
  "geometry": {
    "type": "GeometryCollection",
    "geometries": [{}
```

```

"type": "Point",
"coordinates": [0, 0]
},
{
"type": "LineString",
"coordinates": [[0, 0], [1, 0]]
}
},
"properties": {
"name": "Der Name dieser GeometryCollection"
}
}

```

Einen Anwendungsfall für eine Geometriekollektionen gibt es in der Praxis allerdings nur sehr selten: Meist ist es so, dass jede Geometrie auch eigene Eigenschaften besitzt. Im nächsten Kapitel werden Sie lesen, dass Sie eine Geometrie mit eigenen Eigenschaften im Typ `Feature` beschreiben können und `Feature` Objekte werden mit dem Typ `FeatureCollection` zusammengefasst.

Einer GeoJSON Geometrien Eigenschaften zuordnen

Geometrien sind Formen – nicht mehr und nicht weniger. Sie sind ein zentraler Teil von GeoJSON, aber die meisten Daten, die etwas mit der Welt zu tun haben, sind nicht einfach nur eine Form. Die Formen haben auch eine Identität und Attribute. Ein Polygon stellt beispielsweise den Reichstag dar. Ein anderes Polygone ist die Grenze von Deutschland. Und bei der Arbeit mit den Geometrien ist es wichtig zu wissen, welche Geometrie was ist und, welche Eigenschaften die Geometrie hat. In GeoJSON kann genau dies mit einem Objekt des Typs `Feature` erreicht werden.

Das nachfolgende Programmcodebeispiel enthält ein ganz einfaches `Feature` Element.

```
{
"type": "Feature",
"geometry": {
"type": "Point",

```

```

"coordinates": [0, 0]
},
"properties": {
"name": "Der Name des Punktes"
}
}

```

Das nachfolgende Programmcodebeispiel enthält ein etwas komplexeres Feature.

```

{
"type": "Feature",
"geometry": {
"type": "Polygon",
"coordinates": [[30, 20], [45, 40], [10, 40], [30, 20]]
},
"properties": {
"prop0": "value0",
"prop1": {"this": "that"},
"prop2": true,
"prop3": null,
"prop4": ["wert1", "wert2", "wert3"],
"prop5": 0.0
}
}
```

Eine Eigenschaft mit jedem [JSON-Objekt](#) Datentyp beschrieben werden.

FeatureCollection

So nun haben wir jede Menge Typen kennen gelernt. Den Typ, den Sie sicherlich am meisten nutzen werden, habe ich für den Schluss aufgehoben. Die Syntax einer FeatureCollection können Sie im nachfolgenden Beispiel ablesen.

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Point",  
        "coordinates": [0, 0]  
      },  
      "properties": {  
        "name": "Der Name des Features"  
      }  
    }  
  ]  
}
```

Ein Objekt vom Typ `FeatureCollection` enthält ein Schlüssel-Wert-Paar. Der Wert ist ein Array das aus `Feature` Objekten besteht und der Schlüssel lautet `Features`. Wie der Name schon sagt, darf das Array ausschließlich Objekte vom Typ `Feature` enthalten.

Welche Vorteile bringt ein Objekt vom Typ `FeatureCollection` zusätzlich zu den einzelnen Objekten? Ein Objekt vom Typ `FeatureCollections` ist sehr sinnvoll, wenn Sie mit GeoJSON-Typen arbeiten, die gemeinsame Eigenschaften haben.

todo <http://geojsonlint.com/>

Hinweis:

Sie möchten gerne mit GeoJSON nutzen und haben auch schon die ersten Dateien erstellt. Vielleicht sind Sie auch schon auf das ein oder andere Problem gestoßen oder möchten einfach nur mit der Syntax vertraut werden. Auf der Website <http://geojsonlint.com/> können Sie Ihre GeoJSON Daten testen.

Die Grenzen von GeoJSON

Die Vorteile von GeoJSON hatte ich Ihnen weiter vorne in diesem Kapitel näher gebracht. Wie jedes andere Format hat GeoJSON aber auch seine Grenzen. Diese sind nun Thema dieses Kapitels.

- GeoJSON hat kein Konstrukt das eine Komprimierung unterstützt wie beispielsweise [TopoJSON](#) oder [OSM XML](#).
- GeoJSON unterstützt die gleichen Datentypen wie JSON. JSON unterstützt nicht jeden Datentyp. Zum Beispiel gibt es keinen Typ für Datumswerte in JSON.
- GeoJSON hat kein Konstrukt für die Anzeige von Pop-up Fenstern wie Titel oder Beschreibung. Sie werden aber später sehen, wie Leaflet Sie bei der Erstellung von Pop-up Fenstern unterstützt.
- GeoJSON hat keine Geometrie vom Typ Kreis – oder irgendeine andere Art von Kurve.
- In GeoJSON können Sie den einzelnen Koordinaten – also den Positionen – keine eigene Eigenschaft zuweisen. Wenn Sie die `LineString` Darstellung eines Trainingslaufs haben und Ihr GPS Gerät mehr als 1000 verschiedene Punkte während dieses Laufs zusammen mit Ihrer Herzfrequenz protokolliert hat, bietet GeoJSON keine klare Antwort auf die Frage, wie Sie diese Daten am besten darstellen. Sie könnten eine zusätzliche Eigenschaft als Array mit der gleichen Länge wie das Koordinaten Array speichern – eine klare Regelung gibt es aber nicht.

GeoJson in Leaflet

Leaflet unterstützt alle GeoJSON-Typen. In der Regel werden Sie überwiegend den Typ `Feature` und `FeatureCollection` nutzen. Sie möchten ja sicher nicht nur Geometrie Objekte auf Ihrer Karte anzeigen, sondern auch die Eigenschaften – also weitere Informationen – zu diesen Objekten. Die Typen `Feature` und `FeatureCollection` unterstützt Leaflet besonders gut.

Ein GeoJSON Feature in Leaflet einbinden

Beginnen wir mit einem übersichtlichen Beispiel: Die einfachste Art GeoJSON in Ihrer Karte zu nutzen, ist die Verwendung als Variable direkt – fest programmiert. So etwas sollte man eigentlich nicht machen. Übungsbeispiele lassen sich so aber möglichst einfach darstellen.

Eine andere alternative Art GeoJSON Daten in ein HTML-Dokument einzubinden finden Sie im Kapitel *Choroplethenkarte* und hier im Unterkapitel *Open Data*.

Das nachfolgende Programmcodebeispiel enthält einen Punkt. Sobald Sie diesen – in GeoJSON formatierten – Punkt in einer Variablen gespeichert haben, können Sie diesen ganz einfach zur Karte hinzufügen. Leaflet zeigt auf der Karte als Ergebnis einen Marker an.

Hinweis:

Wir haben festgestellt, dass Leaflet Koordinaten in einer anderen Reihenfolge als GeoJSON schreibt. Wenn Sie die Standardfunktionen in Leaflet verwenden, müssen Sie sich hierüber aber keine Sorgen machen. Leaflet sortiert die Koordinaten selbständig in die passende Form.

Hier also nun das erste praktische Beispiel zum Thema GeoJSON und Leaflet.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var geojsonFeature1 = {
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [7.26469, 50.27264]
  },
}
```

```

"properties": {
  "name": "Gering"
}
};

L.geoJSON(geojsonFeature1).addTo(mymap);

</script>
</body>
</html>

<!--index_973.html-->

```

Was passiert hier genau? Als erste haben wir den GeoJSON Code fest in der Variablen `geojsonFeature1` gespeichert. Als Nächstes haben wir ein Leaflet Objekt vom Typ `GeoJSON` erstellt und diesem unseren GeoJSON Code, also die Variable `geojsonFeature1`, übergeben. Das `GeoJSON` Objekt haben wir gleichzeitig mithilfe der Methode `addTo (mymap)` zum Leaflet Kartenobjekt hinzugefügt. Mehr mussten wir nicht tun. Das Ergebnis ist ein Standard Marker an der Stelle auf der Erde, die das `GeoJSON Point` Element beinhaltet.

Das Objekt [GeoJSON](#) ist ein Leaflet Layer. Ganz konkret erweitert die Klasse `GeoJSON` die Klasse [FeatureGroup](#).

Auf der nachfolgenden Abbildung können Sie sich das Ergebnis ansehen.

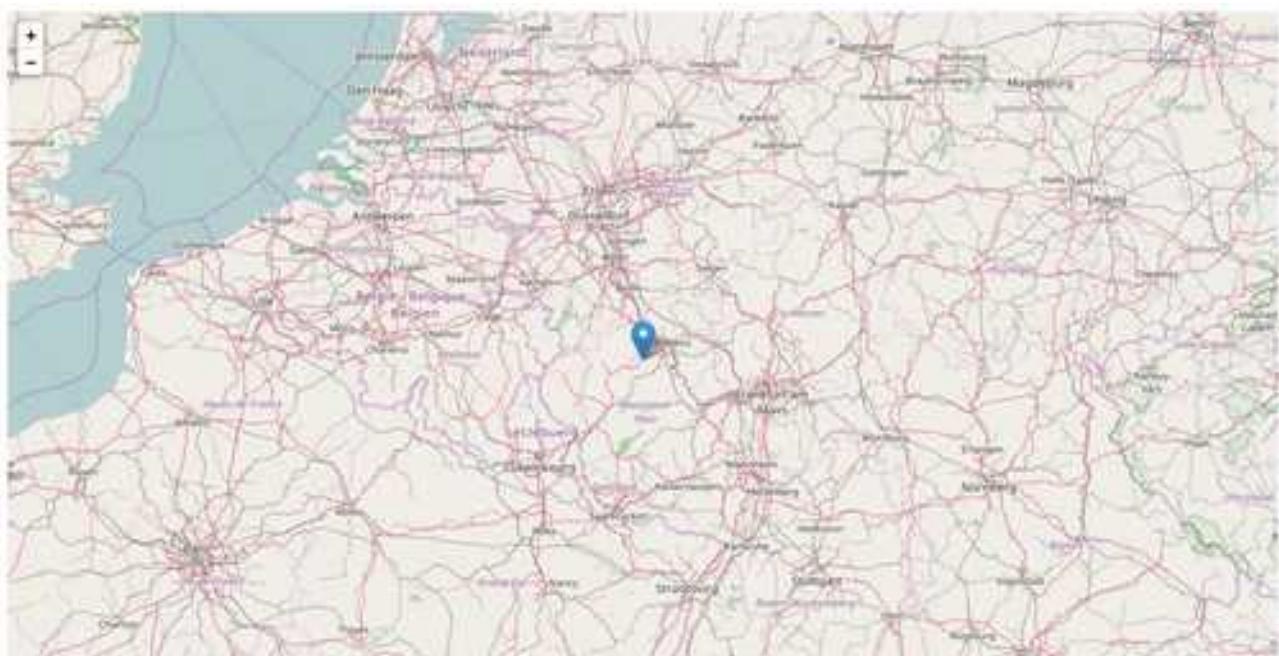


Abbildung 21
981.png

Hinweis:

Wenn Sie an die Zeile

```
L.geoJSON(geojsonFeature1).addTo(mymap);  
noch den Text  
.bindPopup('Pop-up Text');  
anhängen, also  
L.geoJSON(geojsonFeature1).addTo(mymap).bindPopup('Pop-up  
Text');  
schreiben, öffnet sich ein Pop-up Fenster, wenn Sie den Marker anklicken.
```

Ein GeoJSON FeatureCollection in Leaflet einbinden

Das Beispiel im letzten Kapitel enthielt ausschließlich einen Punkt – also ein Feature. Geodaten bestehen in der Regel aus mehreren Geometrien mit dazugehörigen Eigenschaften – aus FeatureCollections. Leaflet liest eine FeatureCollection genauso ein, wie Sie es im letzten Beispiel anhand des Features gesehen haben. Das nächste Beispiel zeigt Ihnen, wie Sie einen Punkt, ein Polygon und eine Linie – wieder mithilfe einer fest kodierten Variablen im GeoJson Format – in einem Schritt auf Ihrer Karte anzeigen könnten.

```
<!DOCTYPE HTML>  
  
<html lang="de">  
  
<head>  
  
<meta charset="utf-8"/>  
  
<title>Eine OSM Karte mit Leaflet</title>  
  
<link rel="stylesheet" href="../leaflet/leaflet.css" />  
  
<script src="../leaflet/leaflet.js"></script>  
  
</head>  
  
<body>  
  
<div style="height: 700px;" id="mapid"></div>  
  
<script>  
  
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);  
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/  
{y}.png').addTo(mymap);  
  
var geojsonFeatureCollection =  
  
{  
  
  "type": "FeatureCollection",
```

```

"features": [
{
  "type": "Feature",
  "geometry": {"type": "Point", "coordinates": [7, 50]},
  "properties": {"prop0": "value0"}
},
{
  "type": "Feature",
  "geometry": { "type": "LineString", "coordinates": [
[7, 50], [7, 51], [6, 51], [6, 52]
]},
  "properties": { "prop0": "value0", "prop1": 0.0 }
},
{
  "type": "Feature",
  "geometry": { "type": "Polygon", "coordinates": [
[ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
]
},
  "properties": { "prop0": "value0", "prop1": {"this": "that"} }
}
]
}

L.geoJSON(geojsonFeatureCollection).addTo(mymap);

</script>
</body>
</html>
<!--index_972.html-->

```

Voila! Drei Elemente mit Standardeigenschaften auf der Leaflet Karte.

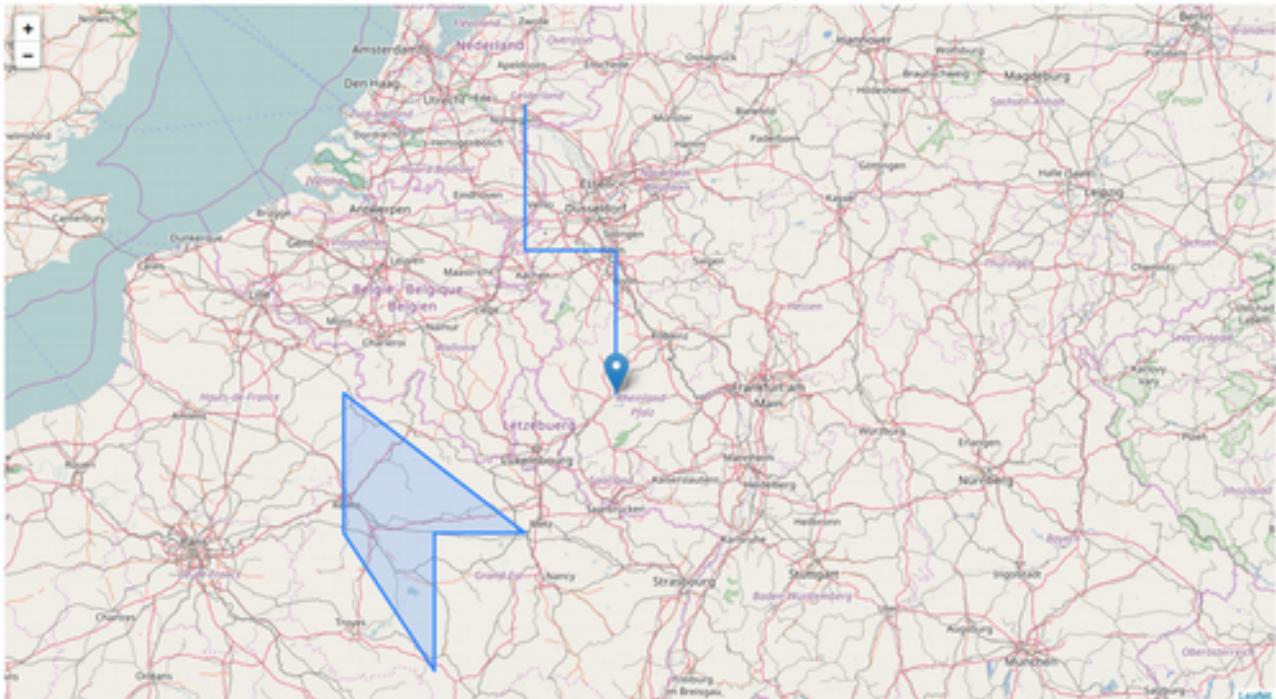


Abbildung 22
961.png

GeoJSON aus leaflet exportieren

So, und nun machen wir genau das Gegenteil. Ein jedes Leaflet Objekt, das wir uns im Kapitel *Die Karte mit Daten bestücken* angesehen haben, hat eine Leaflet Methode mit dem Namen `toGeoJson()`. Und diese Methode tut genau das, was der Name schon vermuten lässt: Das übergebene Leaflet Objekt wird, in ein GeoJSON Objekt umgewandelt und ausgegeben. Sehen Sie sich im nächsten Beispiel die Anwendung der Methode `toGeoJson()` an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
```

```

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var myMarker=L.marker([50.27264, 7.26469]);
var markerAsGeoJSON = myMarker.toGeoJSON();
var geoJsonLayer = L.geoJson().addTo(mymap);
geoJsonLayer.addData(markerAsGeoJSON).bindPopup("Ich bin mit der
Methode .addData() zur Karte hinzugefügt worden. In GeoJson sehe
ich so aus:<br> " + JSON.stringify(markerAsGeoJSON));
</script>
</body>
</html>
<!--index_971.html-->
```

Was zeigt Ihnen dieses Beispiel genau? Das folgende Beispiel zeigt Ihnen, wie Sie einen Marker ins GeoJSON Format konvertieren können. Dazu erstellen Sie zunächst mit `var myMarker=L.marker([50.27264, 7.26469])` einen Leaflet Marker. Danach rufen Sie die Methode `toGeoJSON()` des Markers auf und speichern das zurückgegebene GeoJSON Objekt in der Variablen `markerAsGeoJSON`. Als Nächstes erstellen Sie einen leeren GeoJSON Layer und fügen diesen zum Kartenobjekt hinzu: `var geoJsonLayer = L.geoJson().addTo(mymap)`. Sie hätten den GeoJSON Code in der Variablen `markerAsGeoJSON` wie in vorherigen Beispielen sofort beim Erstellen des Layers als Parameter mitgeben können. Hier wollte ich Ihnen aber die Methode `addData()` zeigen mit der Sie auch nachträglich noch GeoJSON Objekte zur GeoJSON Ebene hinzufügen können.

Im vorausgehenden Beispiel habe ich die Methode `JSON.stringify()` beim Erstellen des Textes im Pop-up Fenster angewandt. Mit der Methode [JSON.stringify\(\)](#) können Sie eine JavaScript Variable in einen Text im JSON-Format konvertieren.

Die nachfolgende Abbildung zeigt das Bild, welches Sie im Browser sehen, wenn Sie die HTML-Datei des vorausgehenden Beispiels im Browser öffnen.

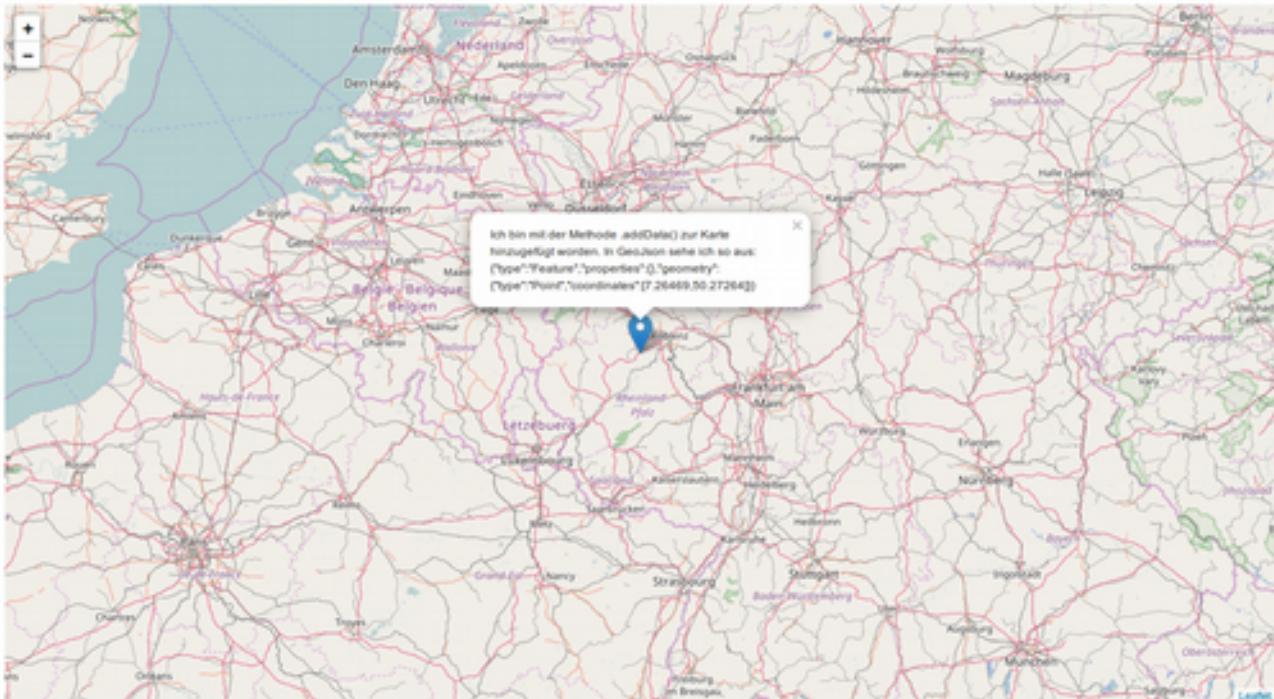


Abbildung 23

960.png

Sie werden nun sicher mein Beispiel als umständlich ansehen. Ich habe einen Marker, denn ich ohne zusätzliche Schritte auf der Karten hätte anzeigen könnte, vorher umgewandelt dann in umgewandelter Form zur Karte hinzugefügt. Diese Vorgehensweise ist nicht Sinn und Zweck der Leaflet Methode `toGeoJSON()`. Sinn und Zweck ist es eher, Daten der Karte zum Export anzubieten.

Stylen

Ein GeoJSON Layer biete Ihnen mit der Methode `setStyle()` die Möglichkeit das Aussehen der Kartenschicht zu gestalten.

Sie können neben den hier beschriebenen Optionen auf eine große Auswahl weitere Stil Optionen zugreifen. Die vollständige Liste finden Sie in der Dokumentation von Leaflet. Sehen Sie sich dazu die Optionen zum Klasse Path an: <http://leafletjs.com/reference-1.1.0.html#path>.

Todo Optionen auflisten?

Beim Erstellen eines GeoJSON Layer einen Stil mitgeben

Der nachfolgende Programmcode zeigt Ihnen, wie Sie Stylesheets beim Erstellen eines GeoJSON Layer als Parameter mitgeben können. Im nachfolgenden Programmcode finden Sie eine Funktion, die je nach GeoJSON Objekt eine andere Farbe zurück gibt. Wenn es sich um den Typ `LineString`

handelt, gibt die Funktion die Farbe Rot zurück, falls ein Polygon vorliegt, antwortet die Funktion mit Violett.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

function styleFunction(feature)
{
  switch (feature.geometry.type)
{
  case 'LineString': return {color: "red"};
  case 'Polygon': return {color: "purple"};
}
}

var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features":
  [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 50]},
      "properties": {"prop0": "value0"}
    },
  ],
}
```

```

{
  "type": "Feature",
  "geometry": { "type": "LineString", "coordinates": [
    [7, 50], [7, 51], [6, 51], [6, 52]
  ] },
  "properties": { "prop0": "value0", "prop1": 0.0 }
},
{ "type": "Feature",
  "geometry": { "type": "Polygon", "coordinates": [
    [ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
  ] },
  "properties": { "prop0": "value0", "prop1": {"this": "that"} }
}
]
}

var geoJsonLayer = L.geoJson(geojsonFeatureCollection,
{style:styleFunction}).addTo(mymap);
</script>
</body>
</html>
<!--index_970.html-->

```

Wenn Sie einen Stil auf alle Objekte anwenden möchten, dann ist es nicht notwendig eine Funktion zu erstellen. Die Zeile
`var geoJsonLayer = L.geoJson(geojsonFeatureCollection, {color: "purple"}).addTo(mymap);`
reicht völlig aus.

Auf der Karte sehen Sie nun die Objekte in der für sie bestimmten Farbe.

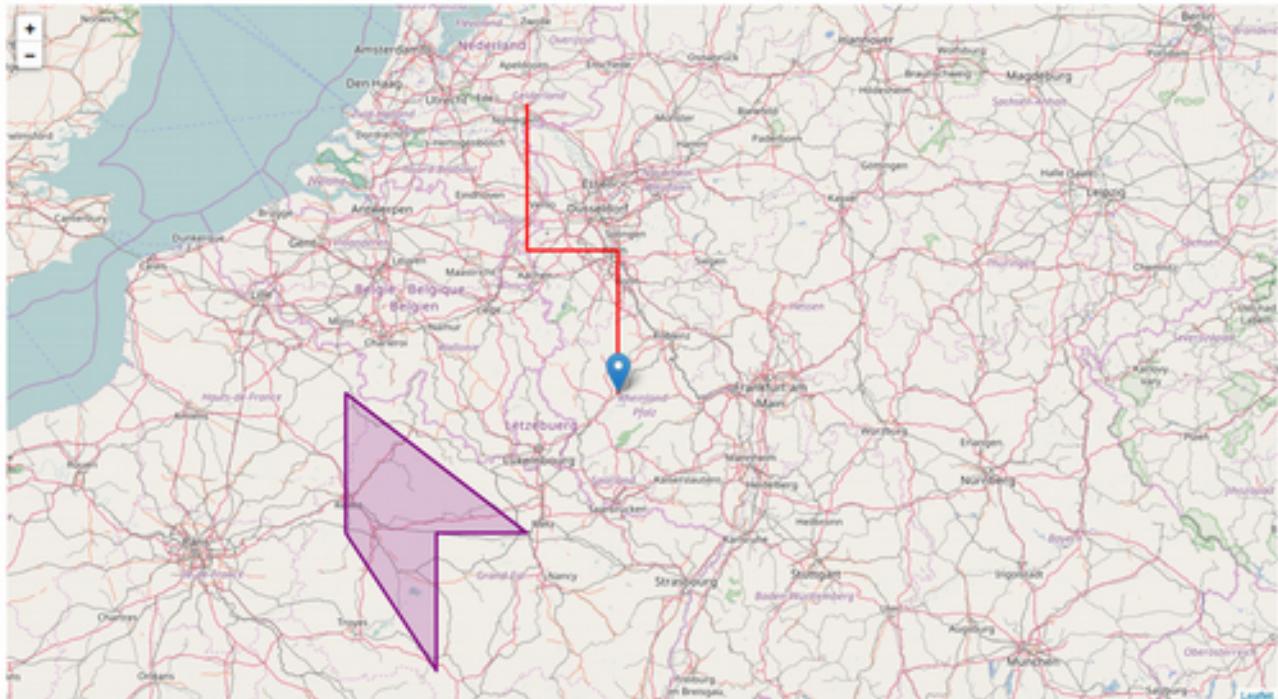


Abbildung 24
959.png

Ein komplexeres Beispiel: Eine andere Farbe beim Überrollen mit der Maus

Im vorherigen Kapitel haben Sie gelernt, wie Sie mit der Option `style` des GeoJSON Layers ein Aussehen festlegen können. Sicherlich ändert sich das Aussehen Ihrer Objekte im Laufe der Zeit. Ganz häufig kommt es vor, dass man Objekte die anklickbar sind und angeklickt wurden, als schon besucht kennzeichnen will. Oder Sie möchten das Objekt, über dem sich die Maus gerade befindet, hervorheben. Genau diese beiden Anwendungsfälle sind Thema im nachfolgenden Beispielcode.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
```

```
<script>

var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

function styleFunction(feature)
{
switch (feature.geometry.type)
{
case 'LineString': return {color: "red"};
case 'Polygon': return {color: "purple"};
}
}

var geojsonFeatureCollection =
{
"type": "FeatureCollection",
"features":
[
{
"type": "Feature",
"geometry": {"type": "Point", "coordinates": [7, 50]},
"properties": {"prop0": "value0"}
},
{
"type": "Feature",
"geometry": { "type": "LineString", "coordinates": [
[7, 50], [7, 51], [6, 51], [6, 52]
] },
"properties": { "prop0": "value0", "prop1": 0.0 }
},
{
"type": "Feature",
"geometry": { "type": "Polygon", "coordinates": [
[ [6, 49], [5, 49], [5, 48], [4, 49], [4, 50] ]
]
},
"properties": { "prop0": "value0", "prop1": {"this": "that"}}
}
```

```

}

]

}

var geoJsonLayer = L.geoJson(geojsonFeatureCollection,
{style:styleFunction}) .addTo(mymap) ;

geoJsonLayer.on('mouseover', styleWhenMouseOver) ;
geoJsonLayer.on('mouseout', styleWhenMouseOut) ;

function styleWhenMouseOver(e) {
geoJsonLayer.setStyle({color:"green"}) ;
}

function styleWhenMouseOut(e) {
geoJsonLayer.eachLayer(function (layer) {
geoJsonLayer.resetStyle(layer) ;
}) ;
}

</script>
</body>
</html>
<!--index_969.html-->

```

Auf den ersten Blick hat sich im Vergleich zum vorherigen Beispiel nichts geändert. Wenn Sie allerdings die Maus über ein Objekt bewegen, sehen Sie eine Änderung. Das Polygon und die Linie verfärben sich nun grün. Der Marker kann seine Farbe nicht ändern. Im Grunde genommen handelt sich bei ihm um ein Image und dieses HTML-Element verfügt nicht über die CSS Eigenschaft `color`. Wenn Sie das Aussehen des Markers ändern möchten, dann müssten Sie diesem eine andere Bilddatei zuordnen.

Wenn Sie ein schon angeklicktes Element mit der Farbe Grau einfärben möchten, dann könnten Sie dies über die Funktion

```

function styleWhenMouseOut(e) {
geoJsonLayer.setStyle({color:"gray"}) ;
})
;
```

```

    }
anstelle von
function styleWhenMouseOut(e) {
geoJsonLayer.eachLayer(function (layer) {
geoJsonLayer.resetStyle(layer);
});
}
erreichen.

```

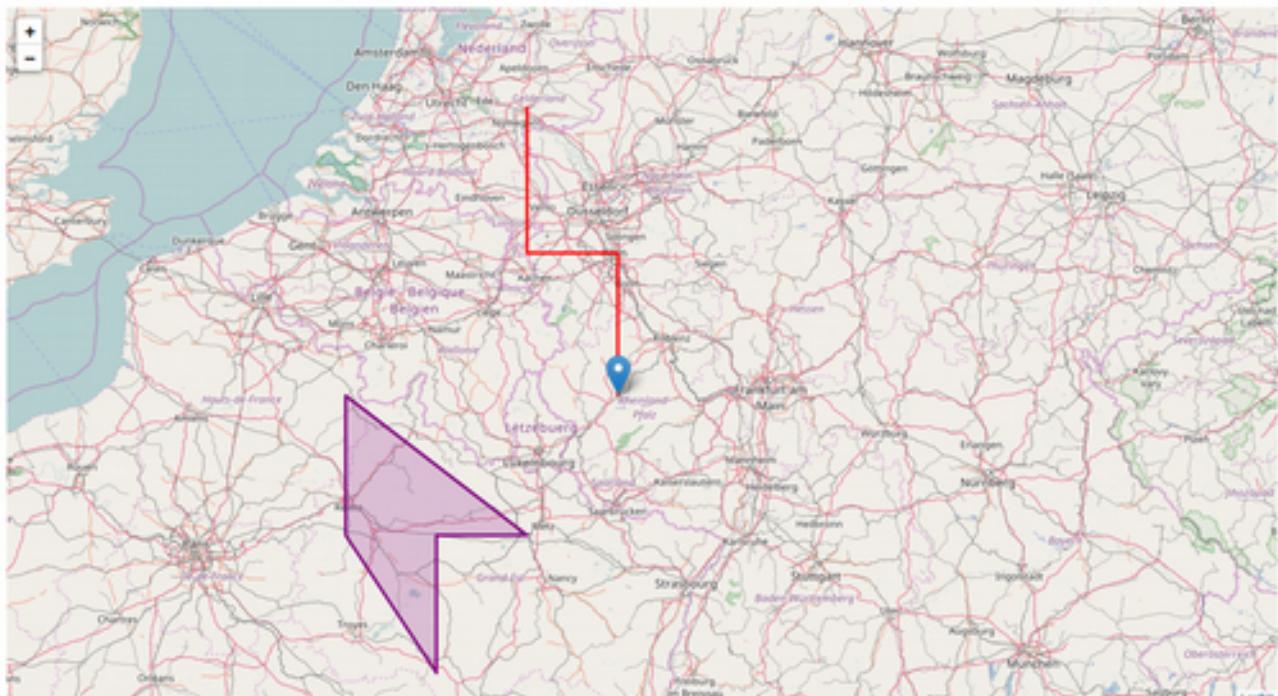


Abbildung 25

958.png

Achtung:

Wenn Sie

```

function styleWhenMouseOut(e) {
geoJsonLayer.eachLayer(function (layer) {
geoJsonLayer.resetStyle(layer);
});
}

```

anstelle von

```

geoJsonLayer.on('mouseout', function(e)
{geoJsonLayer.resetStyle(e.layer);});

```

schreiben würden, würde nur ein Layer – nämlich der, der gerade überrollt wird – geändert.

Iterieren

Interessant werden Karten, wenn Sie viele Informationen bieten. Eine Karte mit vielen Informationen setzt die Arbeit mit vielen Daten für den Kartenersteller voraus. Und, beim Arbeiten mit vielen Daten werden Sie die Möglichkeit, alle Features mit einem Schlag zu bearbeiten, zu schätzen lernen. Iterieren können sie in Leaflet durch GeoJSON Objekte mithilfe der Methode `onEachFeature`.

OnEachFeature – Für jedes Feature

`OnEachFeature` ist eine Methode, die einmal für jedes im Layer vorhandene GeoJSON Objekt vom Typ `Feature` aufgerufen wird. Nützlich ist diese Option zum Anhängen von Ereignissen und/oder Pop-up Fenstern an jedes `Feature` Objekt.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features":
  [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [6, 50]},
      "onEachFeature": function(feature, layer) {
        layer.bindPopup("Hier steht ein Pop-up!");
      }
    }
  ]
}</script>
```

```

"properties": {"name": "Dorf 1"}
},
{
"type": "Feature",
"geometry": {"type": "Point", "coordinates": [7, 50]},
"properties": {"name": "Dorf 2"}
},
{
"type": "Feature",
"geometry": {"type": "Point", "coordinates": [7, 51]},
"properties": {"name": "Dorf 3"}
}
];
var options = {
draggable: true,
title: "Ein Dorf in der Nähe von Gering"
};
L.geoJson(geojsonFeatureCollection, {
onEachFeature: function(feature, layer) {
layer.bindPopup(feature.properties.name);
}
}).addTo(mymap);
</script>
</body>
</html>
<!--index_968.html-->
```

Der vorhergehende Programmcode zeigt Ihnen, wie Sie jedem Feature Objekt, das über die hart kodiert eingefügten GeoJSON Daten eingelesen wird, ein Pop-up anhängen können – und zwar jedem ein individuelles. Der Text für das Pop-up wird ebenfalls aus den GeoJSON Daten ausgelesen, er versteckt sich in der Eigenschaft `feature.properties.name`.

Das nachfolgende Bild stellt die drei Marker dar.

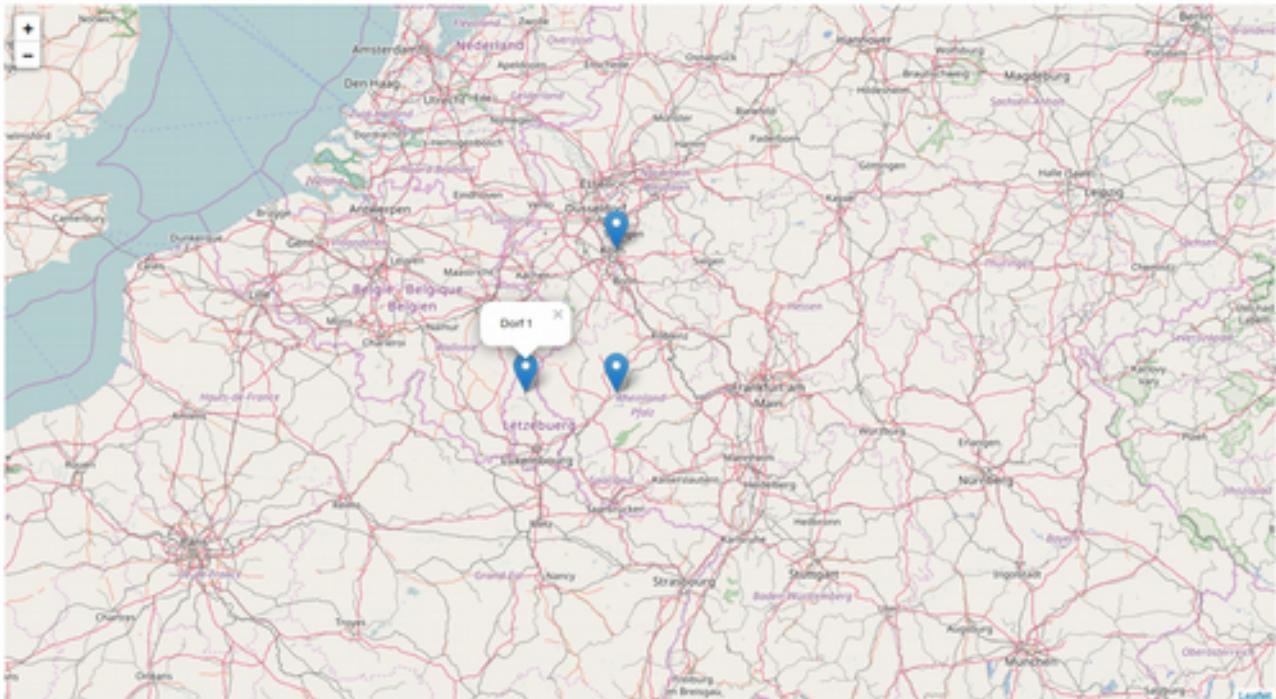


Abbildung 26

.png

Nun würden wir dem Marker aber vielleicht auch gerne ein spezielles Aussehen geben. Vielleicht möchten Sie sogar jeden Marker unterschiedlich gestalten. `onEachFeature()` bietet Ihnen hierzu keine Möglichkeit. Der Marker wird hier automatisch mit Standardwerten erstellt. Leaflet bietet Ihnen aber eine andere Funktion für diesen Zweck. Die Funktion heißt `pointToLayer()` und ein Beispiel dazu, wie Sie mit dieser Methode einen eigenen Marker erstellen und mit individuellen Optionen verstehen können, finden Sie im nächsten Kapitel.

PointToLayer – Punkt zu Ebene

Die Methode `pointToLayer`, die wir uns in diesem Kapitel ansehen, ist spezielle für die Arbeit mit einem GeoJSON Objekten vom Typ `Point` gemacht. Dieses ist das GeoJSON Pendant zum Marker. Wenn wir einen Point beim Erstellen des GeoJSON Layers als Parameter übergeben, dann wird ein Standard Marker erstellt. Wollen wir diesen Marker individuell gestalten, dann brauchen wir entweder einen Variablenamen, den wir ansprechen können, oder wir müssen ihn selbst instanziieren. Für das Instanziieren brauchen wir eine Position oder eine Point. Und nun schließt sich der Kreis. Die Option `pointToLayer` gibt uns Zugriff auf die Koordinaten. Sehen Sie sich das nächste Beispiel an. Ein Beispiel erklärt oft mehr als viele Worte.

```
<!DOCTYPE HTML>
<html lang="de">
```

```
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var geojsonFeatureCollection =
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [6, 50]},
      "properties": {"name": "Dorf 1"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 50]},
      "properties": {"name": "Dorf 2"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [7, 51]},
      "properties": {"name": "Dorf 3"}
    }
  ];
  var options_draggable = {
    draggable: true,
```

```

title: "Ein Ort in der Nähe von Gering"
};

var options_notdraggable = {
draggable: false,
title: "Ein Ort in der Nähe von Gering"
};

L.geoJson(geojsonFeatureCollection, {
pointToLayer: function(feature, latlng) {
switch(feature.properties.name)

{
case "Dorf 1": return L.marker(latlng,
options_draggable).bindPopup(feature.properties.name);
case "Dorf 2": return L.marker(latlng,
options_notdraggable).bindPopup(feature.properties.name);
case "Dorf 3": return L.marker(latlng,
options_notdraggable).bindPopup(feature.properties.name);
}
}
});

}).addTo(mymap);

</script>
</body>
</html>
<!--index_967.html-->

```

Im vorhergehenden Programmcodebeispiel sehen Sie, wie ein Marker erstellt und mit einem individuellen Pop-up Text versehen wird. Das Ergebnis ist auf den ersten Blick genau das Gleiche wie im vorhergehenden Kapitel. Welcher Pop-up Text dem Marker genau zugewiesen wird, ist in dem Beispiel auch von der Eigenschaft `feature.properties.name` abhängig. Zusätzlich werden in diesem Beispiel Optionen von dem Namen abhängig gemacht. Nur der Marker von Dorf 1 kann auf der Karte verschoben werden. Der größte Unterschied ist aber, dass wir den Marker hier selbst erstellen und deshalb Einfluss auf die Optionen haben.

Sehen Sie sich die Karte, die Sie in der folgenden Abbildung sehen in Ihrem Browser an. Auf den ersten Blick hat sich nichts zu dem Beispiel des vorherigen Kapitels geändert. Auf den zweiten Blick werden Sie feststellen, dass Sie nur den Marker von Dorf 1 auf der Karte verschieben können.

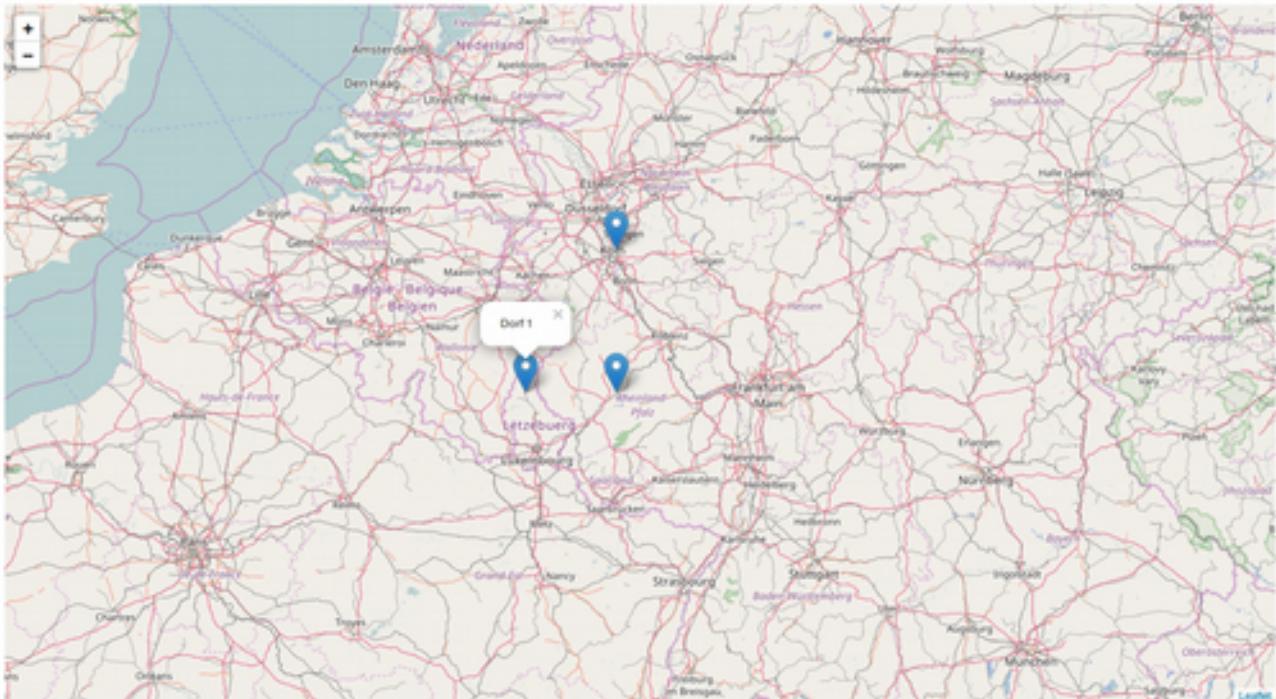


Abbildung 27
956.png

Filtern mit der Option filter

Mithilfe der Option filter können Sie eine große Menge von Daten auf das Wesentliche beschränken. Sehen Sie sich dies im Kleinen im nächsten Beispiel an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var geojsonFeatureCollection =
```

```
{  
  "type": "FeatureCollection",  
  "features":  
  [  
    {  
      "type": "Feature",  
      "geometry": {"type": "Point", "coordinates": [6, 50]},  
      "properties": {"name": "Dorf 1"}  
    },  
    {  
      "type": "Feature",  
      "geometry": {"type": "Point", "coordinates": [7, 50]},  
      "properties": {"name": "Dorf 2"}  
    },  
    {  
      "type": "Feature",  
      "geometry": {"type": "Point", "coordinates": [7, 51]},  
      "properties": {"name": "Dorf 3"}  
    }  
  ]  
};  
  
var options = {  
  draggable: true,  
  title: "Ein Ort in der Nähe von Gering"  
};  
  
L.geoJson(geojsonFeatureCollection, {  
  filter: function(feature, latlng) {  
    switch (feature.properties.name) {  
      case "Dorf 1": return true;  
      default: return false;  
    }  
  }  
}).addTo(mymap);  
</script>  
</body>  
</html>
```

Im Beispiel sehen Sie, dass auf einem GeoJSON Layer nur die Elemente angezeigt werden, deren Rückgabewert beim Filtern positiv oder `true` ist.

Sie nachfolgende Abbildung zeigt es Ihnen: Nur Dorf 1 wird angezeigt. Die beiden anderen Orte, die sich in den GeoJSON Daten befinden, werden heraus gefiltert. Sie sehen nur einen Marker.

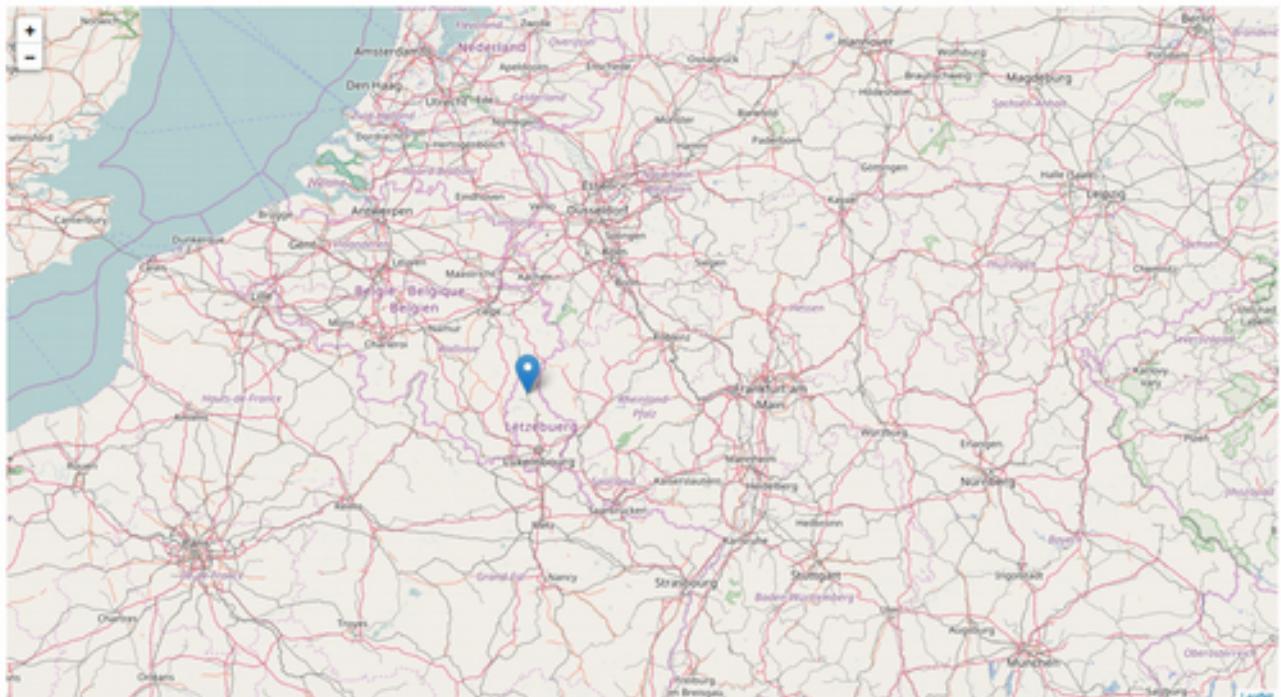


Abbildung 28

955.png

In diesem Kapitel haben wir ...

In diesem Kapitel haben wir uns das Format GeoJSON genau angesehen. Wir haben die GeoJSON Typen mit den Objekten die wir mit Leaflet erstellen können verglichen. Außerdem haben wir viele der Methoden und Optionen, die Leaflet zum Arbeiten mit GeoJSON bietet, angewandt. Eine Karte mit Daten füllen können wir nun. Im nächsten Kapitel sehen wir uns an, wie wir mit den Daten auch Aussagen treffen oder Fragen beantworten können.

Heatmaps und Choroplethenkarten

In den ersten beiden Kapiteln haben wir uns angesehen, wie Sie Elemente entweder direkt – oder mithilfe einer in GeoJSON formatierten Daten – zu Ihrer Karte hinzugefügt können. Jetzt geht es darum, dieses Wissen in die Tat umzusetzen.

In diesem Kapitel werden wir ...

In diesem Kapitel werden wir nicht nur einfach Elemente auf der Karte zeichnen. Wir werden mit der Karte Informationen weitergeben und Fragen beantworten. Insbesondere statistische Daten können auf Karten viel besser veranschaulicht werden, als in einer trockenen Tabelle – und nebenbei macht es sogar Spaß eine solche Karte zu erkunden.

Zwei Typen von Karten – nämlich Heatmaps und Choroplethenkarten – sehen wir uns nun in diesem Kapitel näher an. Beginnen wir mit der Heatmap.

Heatmaps

Heatmaps kennen wir im Deutschen auch unter dem Namen Wärmebild.

Was ist eine Heatmap?

Eine Heatmap ist im Grunde genommen ein Diagramm, mit dem Daten visualisiert werden. Diese Visualisierung dient dazu, in einer großen Datenmenge intuitiv und schnell einen Überblick zu bekommen. In der grafischen Darstellung kristallisieren sich besonders markante Werte oft schnell heraus.

Heatmaps färben Bereich unterschiedlich, wenn die Intensität oder die Dichte des untersuchten Objektes unterschiedlich ist.

Hinweis:

Möchten Sie gerne wissen, wie die Darstellung einer Heatmap technisch umgesetzt wird? Vereinfacht ausgedrückt wird zunächst ein Gitternetz über die Karte gelegt. Basierend auf diesem Gitternetz wird im nächsten Schritt die Anzahl der Punkte in jedem Gitternetz-Bereich gezählt. Je nach Punktanzahl wird zum Schluss die Anzeige angepasst. Ist in einem Bereich kein Punkt vorhanden, wird dieser Bereich nicht mit Farbe gefüllt. Bei den anderen Bereichen wird je nach Anzahl die passende Farbe im Bereich eingeblendet. Dieses Verfahren nennt man Multivariate Kerndichte Schätzung (englisch: Multivariate kernel density estimation). Detaillierter können Sie dies beispielsweise bei [Wikipedia](#) nachlesen.

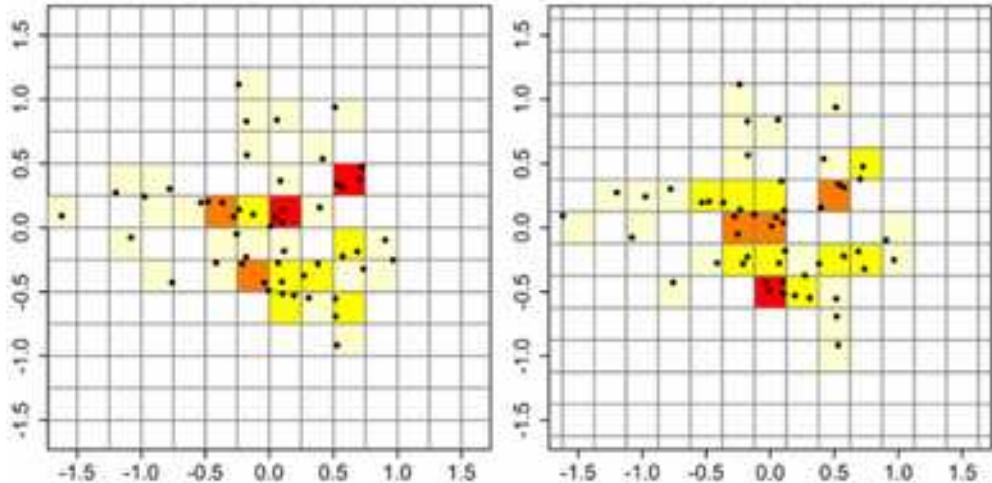


Abbildung 29: (By Drleft (Own work) [CC BY-SA 3.0] (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>]), via Wikimedia Commons)

Meist werden bei geringer Intensität oder geringer Dichte kalte Farben verwendet und bei einem hohen Aufkommen wird der Bereich mit warmen Farben eingefärbt. Dies erklärt auch den Namen Heatmap – der englische Begriff für Hitze ist *heat*.

Hinweis:

Blau gilt als kalte Farbe, Rot, Orange und Gelb gelten als warme Farben.

Heatmaps in Leaflet – Dichte

Unser erstes Beispiel zeigt eine Heatmap, die die unterschiedliche Dichte von Punkten sichtbar macht. Die Funktionen zum Anzeigen einer Heatmap werden nicht im Standardumfang von Leaflet mitgeliefert. Es handelt sich hierbei um eine besondere Funktion, die im Normalfall nicht zur Darstellung einer digitalen Karte benötigt wird. Leaflet selbst konzentriert sich auf die Anwendungsfälle, die üblicherweise beim Anzeigen einer Karte benötigt werden – ist aber offen für Plugins. Eines dieser Plugins ist [Leaflet.heat](#).

Weiter Plugins, mit denen Sie Wärmeabbildungen oder ähnliche Visualisierungen aus Vektordaten erstellen können, finden Sie auf der Website von Leaflet im Bereich Plugins:
<http://leafletjs.com/plugins.html#heatmaps>

[Nachfolgende](#) finden Sie das erste Beispiel für diesen Themenbereich.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.1555 , 7.591838], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var points =
[50.1555 , 7.591838 , 0.2],
[50.0931 , 7.664177 , 0.2],
...
[50.088041 , 7.652033 , 0.2],
[50.088041 , 7.652033 , 0.2],
[50.088041 , 7.652033 , 0.2]
];
var heat = L.heatLayer(points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1,
radius: 15,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);
</script>
</body>
```

```
</html>  
<!--index_965.html-->
```

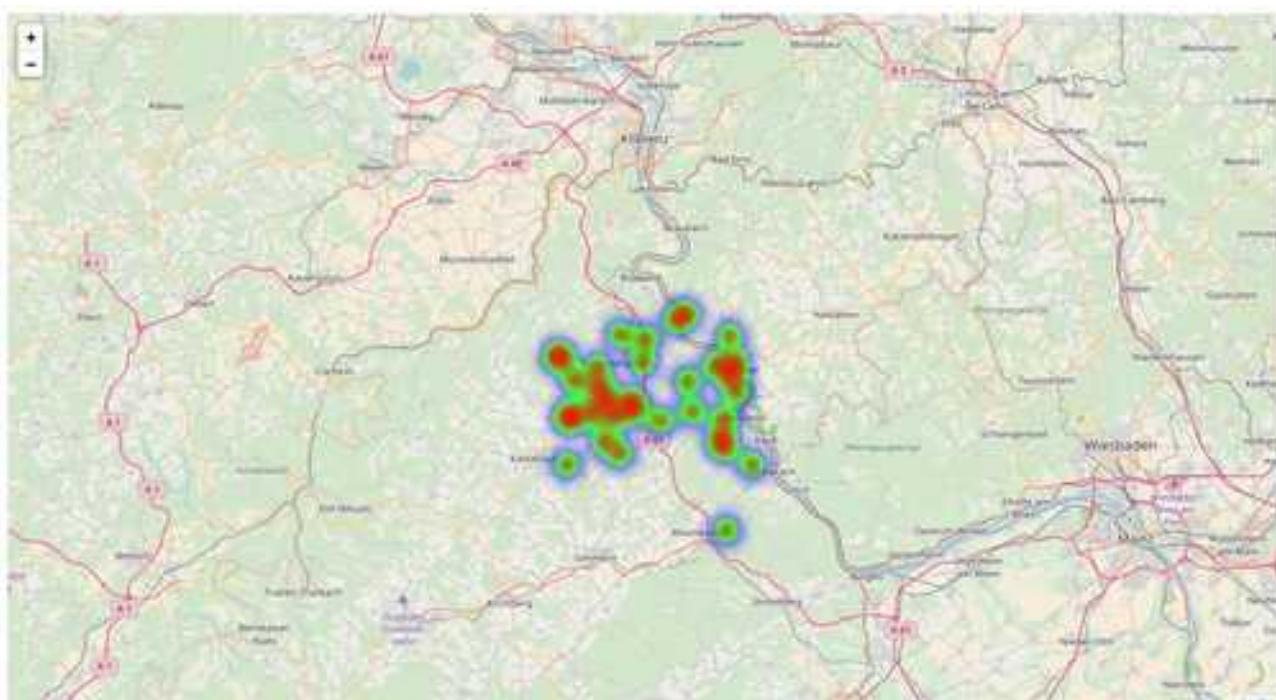
Was haben wir gemacht? Als Erstes haben wir die Skript-Datei zum Plugin mit `<script src="leaflet-heat.js"></script>` eingebunden. Ich hatte die Datei von der oben verlinkten Website kopiert und unter dem Namen `leaflet-heat.js` im Verzeichnis der HTML-Datei abgelegt. Danach habe ich eine Variable mit dem Namen `points` und dem Typ Array erstellt, in der 700 Koordinaten gespeichert sind. Zu guter Letzt habe ich ein Objekt der Klasse `L.heatLayer` instanziert und diesem die Punkte in der Variablen `points` und einige Optionen übergeben.

Der Dritte Wert bei der Übergabe der Daten steht für die Intensität.

```
var points = [  
  [50.1555 , 7.591838 , 0.2],  
  ...
```

Ich habe diesen Wert hier absichtlich immer mit der gleichen Zahl belegt. Ich nutze dieses Plugin nicht für die Visualisierung der Intensität. Hierfür zeige ich Ihnen im nächsten Kapitel eine andere Leaflet Erweiterung.

Das Ergebnis sehen Sie im nachfolgenden Bild. Da wir noch nichts gestaltet haben, sehen Sie Standardansicht. Diese ist noch sehr rudimentär. Das geht besser, Sie werden sehen.



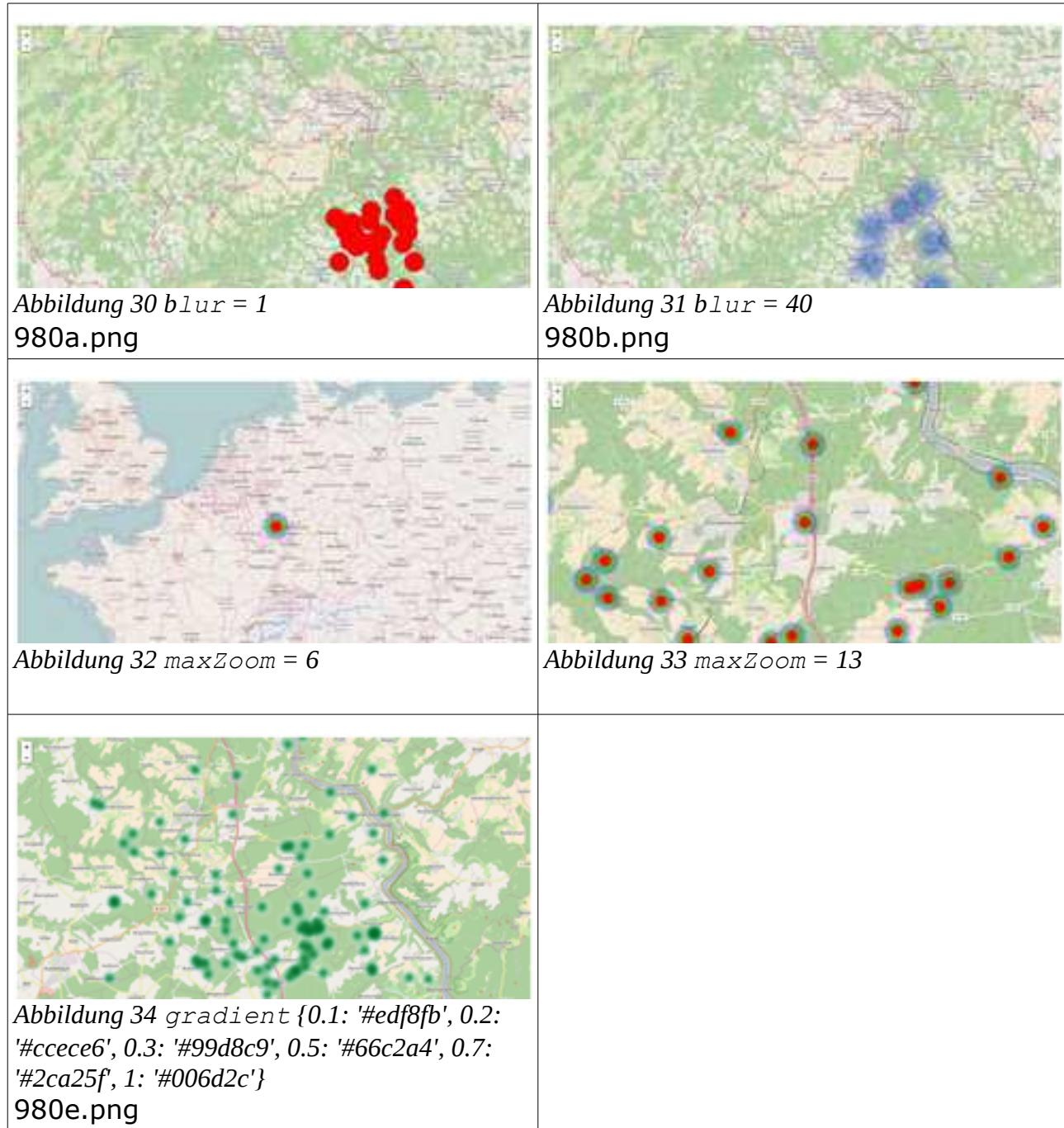
Stile-Optionen

Das [Plugin Leaflet.heat](#) erlaubt es Ihnen Parameter zu übergeben. Dabei haben Sie folgende Möglichkeiten:

- `minOpacity`:
`minOpacity` gibt die minimale Dichte an, ab der die Anzeige beginnt.
- `maxZoom`:
`maxZoom` sollte auf die Zoomstufe gesetzt werden, bei der die Punkte die maximale Intensität erreichen. Als Standard wird der Wert, der für `maxZoom` in der Karte gesetzt ist, verwendet. Auf die Darstellung des Heatmap-Layers hat dies keine Auswirkungen. Sie sollten den Wert so setzen, dass Ihre Karte die beste Aussagekraft hat. Wenn der Wert zu hoch ist, kann es sein, dass die Punkte so weit auseinander liegen, dass der dichteste Bereich nicht mehr deutlich erkennbar ist. Setzen Sie Wert zu niedrig an, kann man die Daten vielleicht nicht mehr zusammenhängend im Überblick sehen.
- `max`:
Maximale Punktintensität – 1.0 ist der Standardwert. Diese Option arbeitet momentan nicht [korrekt](#). Aus diesem Grund stelle ich Ihnen im nächsten Kapitel ein alternatives Plugin zur Visualisierung der Intensität vor.
- `radius`:
Dieser Wert erklärt sich meiner Meinung nach von selbst. Mit dem Wert `radius` geben Sie die Größe an, in der die Punkte angezeigt werden sollen. 25 ist der Standardwert.
- `blur`:
Mit dem Wert `blur` können Sie die Schärfe beeinflussen. Der Wert bestimmt, wie viele Punkte zusammen gefasst werden. Ein niedriger `blur` Wert erzeugt einzelnen Punkte, wohingegen ein hoher Wert mehrere Punkte zusammenfasst. Standardmäßig ist der Wert auf 15 festgesetzt.
- `Gradient`:
Der Wert `Gradient` steht für die Konfiguration des Farbverlaufs. Standardwert ist `{0.4: 'blue', 0.65: 'lime', 1: 'red'}`. Sie können beliebig viele Werte zwischen 0 und 1 angeben. Die Farbe die am Rand, also bei geringer Dichte angezeigt werden soll, sollten Sie mit dem

niedrigen Wert angeben. Die Farbe, die im Zentrum zu sehen sein soll, geben Sie idealerweise beim Wert 1 an.

Vergleich einer Ansicht mit unterschiedlichen Werten für die Optionen `blur` und `maxZoom`



Gefallen Ihnen die kalten und warmen Farben nicht? Möchten Sie lieber Ihre eigene Farbzusammenstellung nutzen? Die Website <http://colorbrewer2.org> hilft beim Auswählen von Farben.

Methoden

Zusätzlich zu den Stylesheet Optionen bietet Ihnen das Plugin Leaflet.heat vier Methoden:

- `setOptions(options)` :
Diese Methode setzt die Optionen der Heatmap neu und zeichnet die Heatmap neu.
- `addLatLng(latlng)` :
Diese Methode fügt einen Punkt zu Heatmap hinzu und aktualisiert die Ansicht gleichzeitig.
- `setLatLngs(latlngs)` :
Diese Methode liest die Daten neu ein und aktualisiert danach die Ansicht.
- `Redraw()` :
Diese Methode zeichnet die Heatmap neu. Sie beim Ausführen der drei anderen Methoden automatisch ausgeführt, so dass Sie dies nicht selbst veranlassen müssen.

Die Methode `AddLatLng()`

Mithilfe der Methode `AddLatLng()` können Sie nachträglich noch einen Punkt zum Heatmap Layer hinzufügen.

```
&lt;!DOCTYPE HTML&gt;
&lt;html lang="de"&gt;
&lt;head&gt;
&lt;meta charset="utf-8"/&gt;
&lt;title&gt;Eine OSM Karte mit Leaflet&lt;/title&gt;
&lt;link rel="stylesheet" href="../leaflet/leaflet.css" /&gt;
&lt;script src="../leaflet/leaflet.js"&gt;&lt;/script&gt;
&lt;script src="leaflet-heat.js"&gt;&lt;/script&gt;
&lt;/head&gt;
&lt;body&gt;
&lt;div style="height: 700px;" id="mapid"&gt;&lt;/div&gt;
&lt;script&gt;
var mymap = L.map('mapid').setView([50.27264, 7.26469], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
```

```

var points = [
[50.1555, 7.591838, 0.2],
..
[50.188041, 7.662033, 0.2],
[50.88041, 7.52033, 0.2]
];
var heat = L.heatLayer(points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);
heat.addLatLng([50.27264, 7.26469, 0.2]);
</script>
</body>
</html>
<!--index_964.html-->

```

In der nächsten Abbildung können Sie den neu hinzugefügten Punkt erkennen.

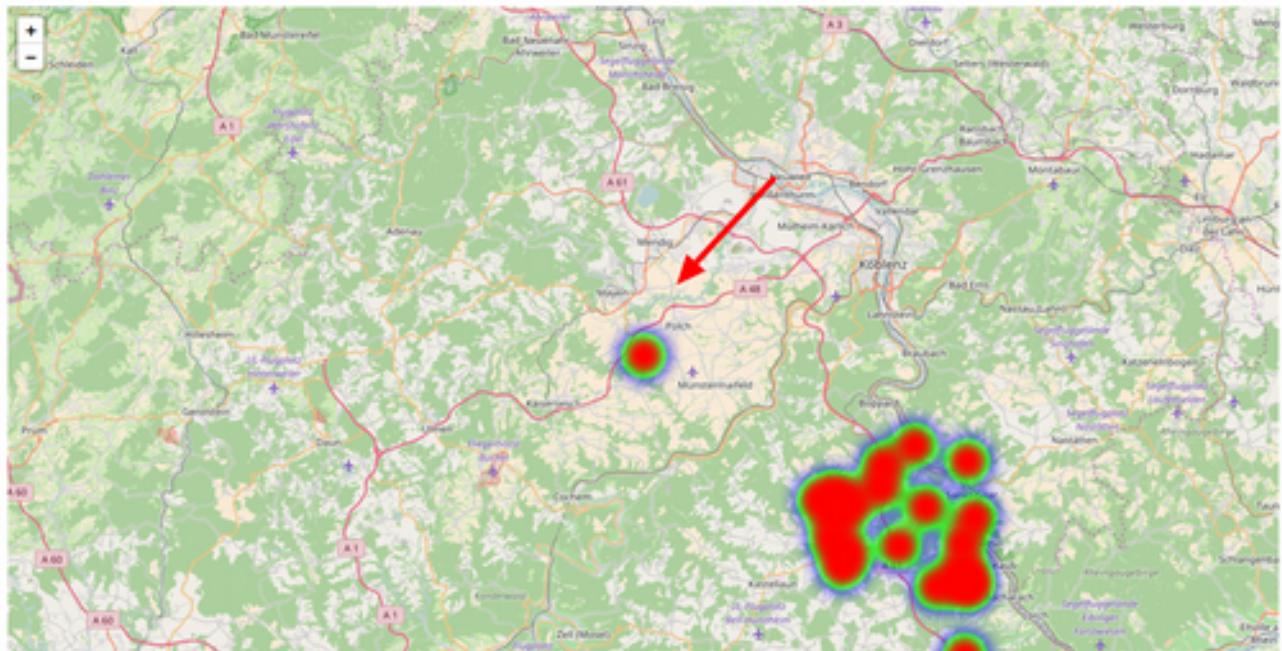


Abbildung 35

Möchten Sie es Website Besuchern ermöglichen Punkte zur Heatmap auf ihrer Karte hinzuzufügen? Eine Demo auf der Website zum Plugin zeigt genau solch ein als Beispiel:
<http://leaflet.github.io/Leaflet.heat/demo/draw.html>

Die Methoden `addLatLng()`, `addLatLngs()` und `setOptions()` in einem Beispiel

In diesem Kapitel stelle ich Ihnen ein Beispiel vor, dass per Schaltfläche die verschiedenen Methoden anwendet.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<button onclick="add()">Punkt hinzufügen</button><br>
<button onclick="newPoint()">Daten neu setzen</button><br>
<button onclick="reset()">Reset</button><br>
<button onclick="setOptions()">Farben ändern</button>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.27264, 7.26469], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var points = [
[50.1555, 7.591838, 0.2],
..
[50.088041, 7.652033, 0.2],
[50.088041, 7.652033, 0.2]
];
```

```

var heat = L.heatLayer(points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);

function add()
{
heat.addLatLng([50.1, 7.1, 0.2]);
}

function newPoint()
{
heat.setLatLngs([[50.5, 7, 0.2]]);
}

function reset()
{
heat.setLatLngs(points);
}

function setOptions()
{
heat.setOptions({gradient: {0.4: 'black', 0.65: 'gray', 1: 'white'}
});
}

```

</script>

</body>

</html>

<!--index_963.html-->

Was passiert genau in diesem Beispiel? Zunächst fügen wir vier Schaltflächen in das HTML-Dokument ein. Eine Schaltfläche führen eine der Methoden `add()`, `newPoint()`, `reset()` oder `setOptions()` aus, wenn sie angeklickt wird. Der Aufruf von `add()` führt den Code `heat.addLatLng([50.1, 7.1, 0.2]); aus`

fügt dabei den Punkt [50.1, 7.1, 0.2] zu den Heatmap Daten hinzu. Die Methode `newPoint()` führt die Programmcodezeile `heat.setLatLngs([[50.5, 7, 0.2]])`; aus und setzt dabei die Daten des Heatmap Layers neu. Diese enthalten nun den einen Punkt im Array [[50.5, 7, 0.2]]. Die Methode `reset()` führt den Code `heat.setLatLngs(points)`; aus und setzt damit die Daten des Heatmap Layers wieder mit den Daten der Variablen `points`. Achtung: Wenn Sie einen Punkt zum Layer hinzugefügt haben, als die Variable `points` Datenbestand des Layers war, dann sind diese Daten nun auch in der Variablen `points` enthalten! Die Methode `setOptions()` ist meiner Meinung nach selbsterklärend. Sie ändert einfach nur CSS Eigenschaften des Layers.

Einen Screenshot der Karte aus diesem Beispiel sehen Sie in der nachfolgenden Abbildung.

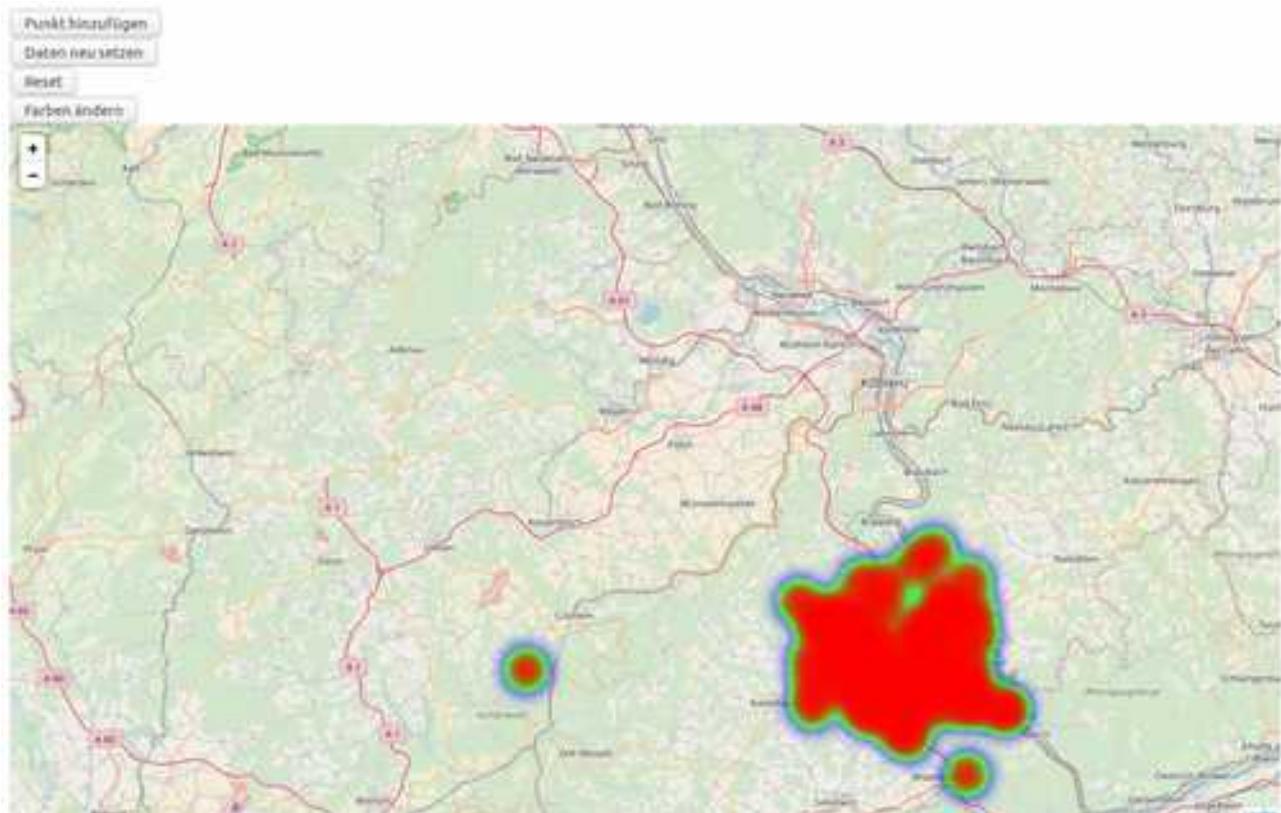


Abbildung 36
953.png

Marker

In nächsten Beispiel sehen Sie, wie Sie jedem Punkt in einer Heatmap einen Marker hinzufügen können.

```
<!DOCTYPE HTML>
```

```
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid').setView([50.1555 , 7.591838], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var points = [
[50.1555 , 7.591838 , "<img
src='http://lorempixel.com/200/200/'>"] ,
..
[50.088041 , 7.652033 , "<img
src='http://lorempixel.com/200/200/'>"]
];

for(var i=0;i<points.length;i++)
{
L.marker(
[parseFloat(points[i][0]),parseFloat(points[i][1])],
{opacity:0}).bindPopup(points[i][2],
{keepInView:true}).addTo(mymap);
}

var heat = L.heatLayer(points,
{
minOpacity: 0.95,
blur: 15,
maxZoom: 15,
max: 1,
```

```

radius: 15,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);
</script>
</body>
</html>
<!--index_962.html-->

```

Was ist diesem Beispiel anders? Wir haben der Variablen point einen dritten Wert mitgegeben. Hier die Adresse zu einem Bild. Dies hätte aber auch ein gewöhnlicher Text sein können. Diesen dritten Wert haben wir dann in der Methode bindPopup(**points[i][2]**, {keepInView}) dazu genutzt, einen individuellen Marker zu kreieren.

So wie in der nächsten Abbildung könnte die fertige Karte aussehen.

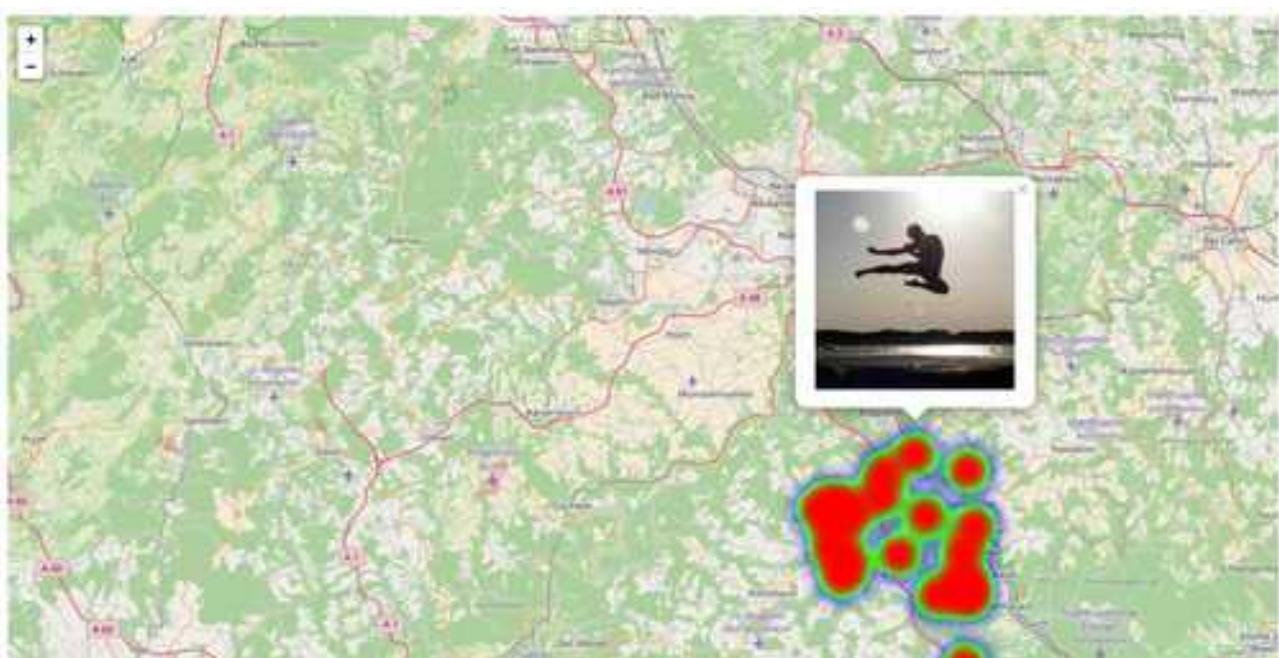


Abbildung 37
980.png

Heatmaps mit Leaflet – heatmap.js – Intensität

Im vorhergehenden Kapitel haben wir eine Heatmap mithilfe des Plugins Leaflet.heat erstellt. Diese Heatmap hat die Dichte von Punkten auf einer Karte visualisierte. Wir haben bisher die Möglichkeit die Intensität zu visualisieren nicht genutzt. Ich hatte Ihnen erklärt, warum ich Ihnen lieber ein anderes Plugin für diesen Zweck vorstellen möchte. Als Nächstes möchte ich nun mit Ihnen eine Heatmap, die die Intensität der Eigenschaft eines Punktes darstellt,

mithilfe des Plugins [heatmap.js](#) erarbeiten. Herunterladen können Sie das Plugin unter der Adresse <https://www.patrick-wied.at/static/heatmapjs/plugin-leaflet-layer.html> und die Dokumentation finden sie unter <https://www.patrick-wied.at/static/heatmapjs/docs.html>.

Nachfolgende sehen Sie das erste Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="heatmap.js"></script>
<script src="leaflet-heatmap.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var myData={
  max: 100,
  data: [
    {lat: 51.0934, lon:8.666819, value: 99},
    ..
    {lat: 50.088041, lon:7.652033, value: 23},
    {lat: 50.088041, lon:7.652033, value: 23}]
};

var baseLayer = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png');

var cfg = {
  "radius": .1,
  "maxOpacity": .5,
  "scaleRadius": true,
  "useLocalExtrema": true,
  latField: 'lat',
```

```

    lngField: 'lon',
    valueField: 'value'
};

var heatmapLayer = new HeatmapOverlay(cfg);

heatmapLayer.setData(myData);

var map = new L.Map('mapid', {
center: new L.LatLng(50.0586, 7.6568),
zoom: 8,
layers: [baseLayer, heatmapLayer]
});
</script>
</body>
</html>
<!--index_961.html-->

```

Was zeigt Ihnen dieses Beispiel genau? Zunächst einmal müssen Sie neben dem Skript leaflet.heat.js auch das Skript heat.js einbinden. Als Nächstes müssen Sie die Daten, die Sie visualisieren möchten, angeben – diese müssen eine Angabe zu dem maximal möglichen Wert enthalten: var myData={ max: 100,data: [{lat: 51.0934, lon:8.666819, value: 99}.... Danach können Sie den Heatmap Layer mit den Optionen cfg erstellen. Der Eintrag hierfür lautet var heatmapLayer = new HeatmapOverlay(cfg);. Diesem Layer können Sie nun die Daten mit heatmapLayer.setData(myData); hinzufügen. Zum Schluss können diesen Layer nun zum Kartenobjekt hinzufügen:
L.Map('mapid', { center: new L.LatLng(50.0586, 7.6568), zoom: 8, layers: [baseLayer, heatmapLayer]});

Wie das aussehen sollte, können Sie sich in der nächsten Abbildung ansehen.

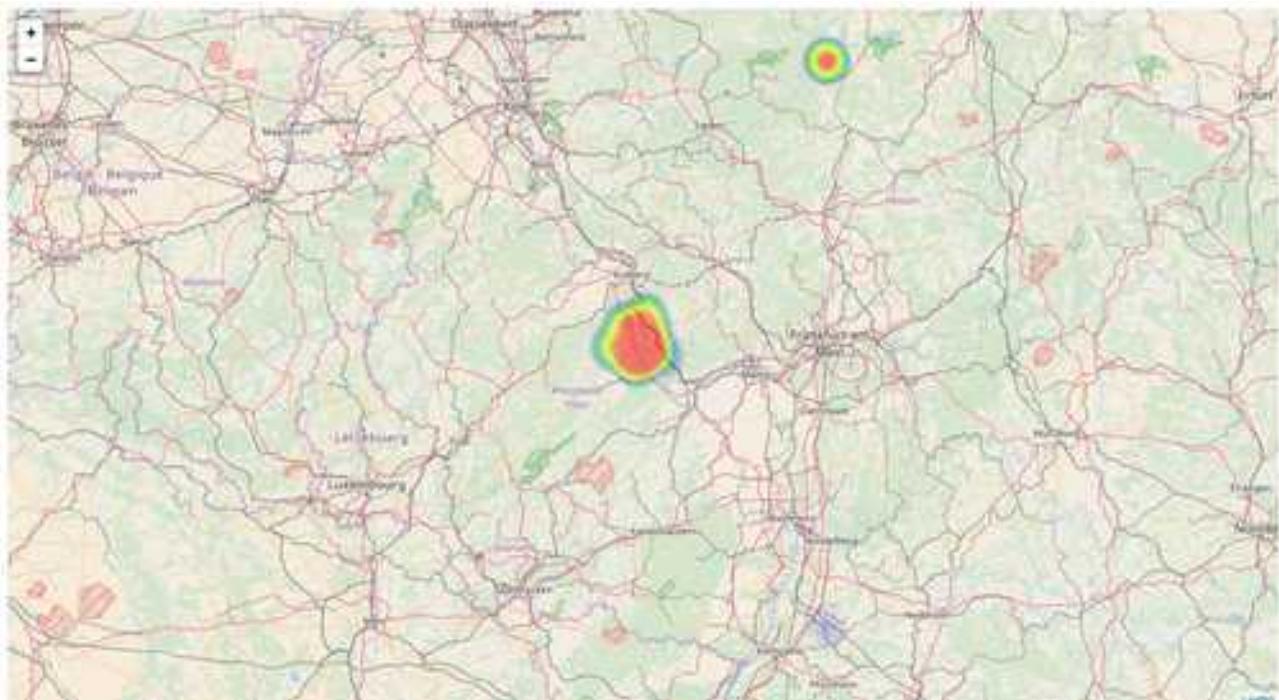


Abbildung 38
952.png

Dokumentation und Methoden

Das nachfolgende Beispiel zeigt Ihnen, wie unterschiedlich Punkte mit unterschiedlicher Intensität aussehen können.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="heatmap.js"></script>
<script src="leaflet-heatmap.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var myData={

max: 100,
data: [
{lat: 51.0934, lon:8.666819, value: 99},
```

```
...
{lat: 50.088041, lon:7.652033, value: 23}]
};

var baseLayer = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png');

var cfg = {
"radius": .1,
"maxOpacity": .5,
"scaleRadius": true,
"useLocalExtrema": true,
latField: 'lat',
lngField: 'lon',
valueField: 'value',
gradient: {
'.4': 'blue',
'.8': 'lime',
'.95': 'red'
},
blur: 0.75
};

var heatmapLayer = new HeatmapOverlay(cfg);

var map = new L.Map('mapid', {
center: new L.LatLng(50.0586, 7.6568),
zoom: 8,
layers: [baseLayer, heatmapLayer]
});

heatmapLayer.setData(myData);

var test1 = {lat:51,lon:6,value:99};
var test2 = {lat:50.5,lon:6,value:60};
var test3 = {lat:50,lon:6,value:40};
var test4 = {lat:49.5,lon:6,value:20};

heatmapLayer.addData(test1);
heatmapLayer.addData(test2);
heatmapLayer.addData(test3);
heatmapLayer.addData(test4);

</script>
```

```
</body>
</html>
<!--index_960.html-->
```

Im vorhergehen Codebeispiel haben wir 4 Punkte mit unterschiedlicher Intensität – 99, 60, 40 und 20 bei einem maximalen Wert von 100 – zur Karte hinzugefügt. In der nachfolgenden Abbildung sehen Sie die unterschiedliche Darstellung auf der Karte.

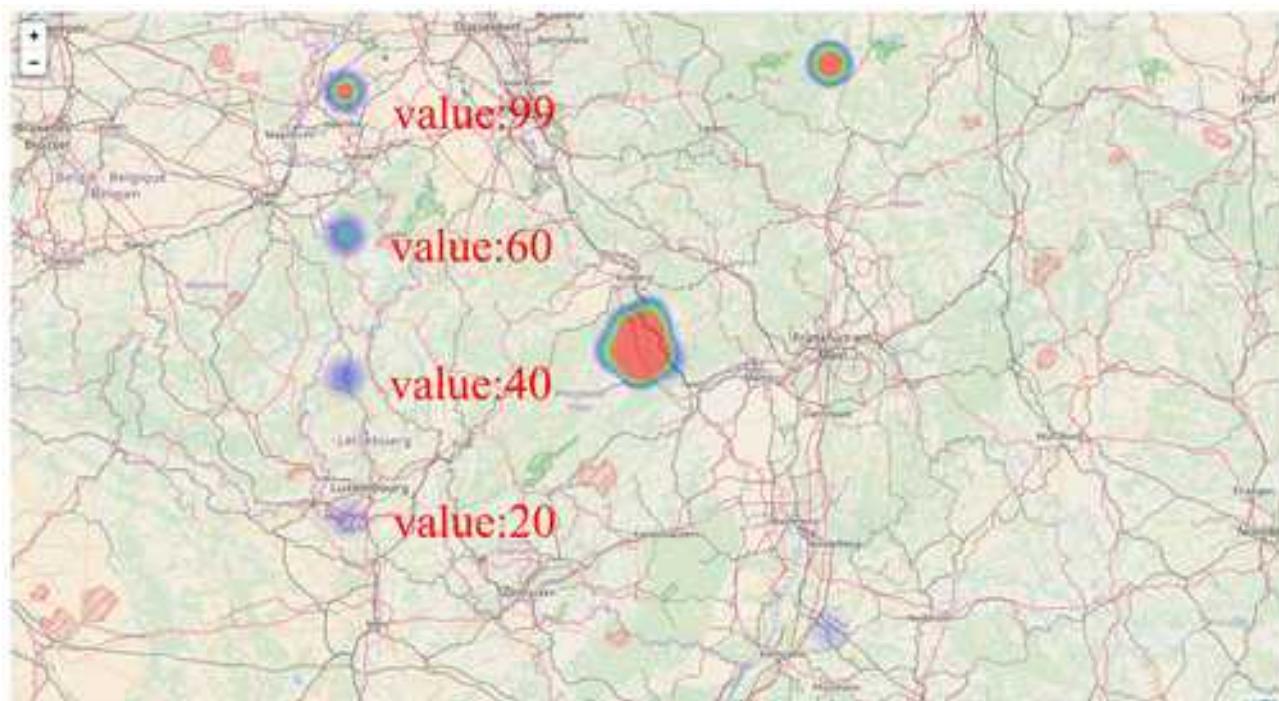


Abbildung 39
949.png

Interaktive Heatmaps

Auch auf eine Heatmap können Sie Benutzer interaktiv Daten hinzufügen lassen. Wie das geht sehen Sie im nachfolgenden Beispiel.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
```

```

<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.1555, 7.591838], 15);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var points = [];
var heat = L.heatLayer(points,
{
blur: 15,
maxZoom: 15,
max: 1.0,
radius: 25,
gradient: {0.4: 'blue', 0.65: 'lime', 1: 'red'}
}).addTo(mymap);

function addpoint(e){
heat.addLatLng(e.latlng);
}

mymap.on('click', addpoint);
</script>
</body>
</html>
<!--index_959.html-->

```

959.html

Was haben wir gemacht? Als Erstes haben wir mit der Zeile `<script src="leaflet-heat.js"></script>` das Plugin leaflet.heat eingebunden. Wichtig ist, dass ein Doppelklick keine Zoom-Stufen Änderung auf der Karte auslöst. So stellen wir sicher, dass jemand nicht versehentlich zu schnell klickt und ungewollt die Zoom-Stufe ändert, obwohl er eigentlich zwei Punkte hinzufügen will. Dies haben wir verhindert, in dem wir die Option `doubleClickZoom auf false setzen`. Zu Beginn sollen noch keine Daten auf

der Karte angezeigt werden. Deshalb haben wir zunächst einen leeren Datensatz mit `var points = [];` erstellt und diesen dann einem `L.heatLayer` Objekt bei der Instanziierung als Parameter mitgegeben `var heat = L.heatLayer(points, {...});`. Nun fehlt noch die Methode, in der ein Punkt hinzugefügt wird. Die ist mit `function addpoint(e)`
`{ heat.addLatLng(e.latlng); }` schnell erstellt. Und `mymap.on('click', addpoint);` bewirkt, dass diese auch ausgeführt wird, wenn jemand auf die Karte klickt.

Das Ergebnis sehen Sie im Browser, wenn Sie die HTML Datei dieses Beispiels öffnen. Zunächst wird die Karte ganz normal angezeigt. Sie sehen keinen Wärmepunkt. Wenn Sie mit der Maus auf einen Punkt auf der Karte klicken, wird an dieser Stelle ein Wärmepunkt hinzugefügt.

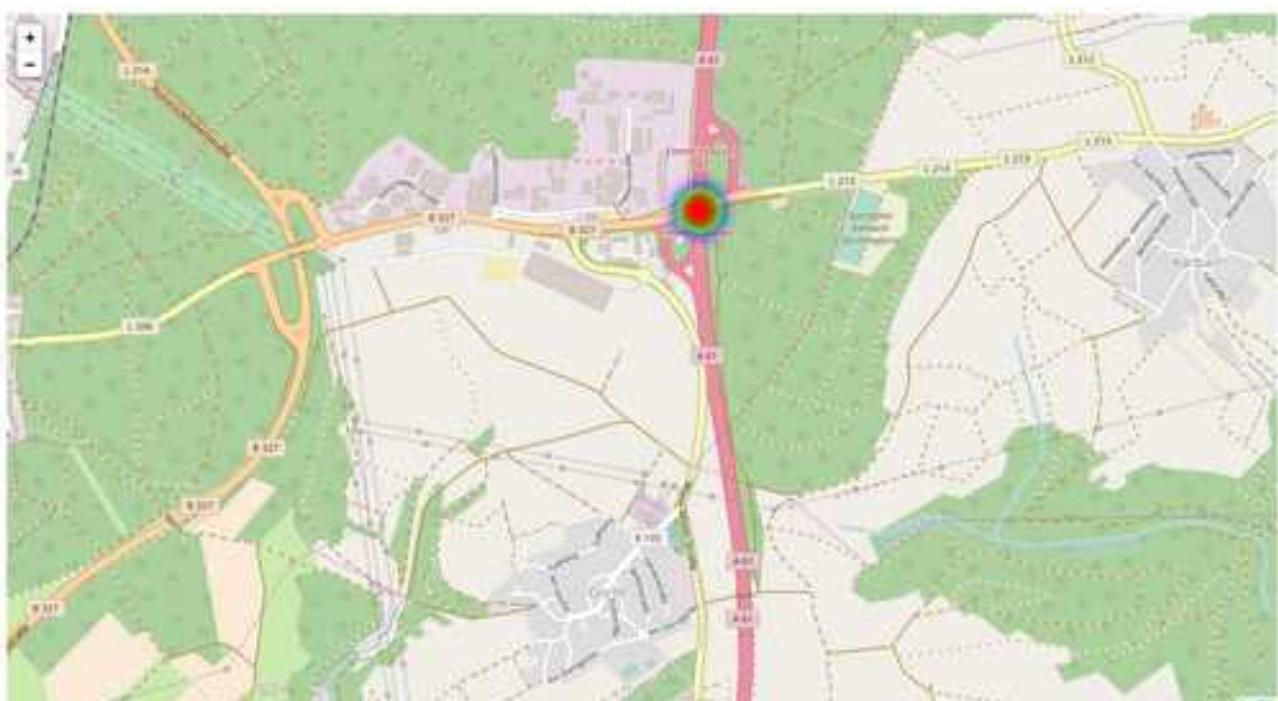


Abbildung 40

951.png

Animierte Heatmaps

Das ist fast wie Kino – im nächsten Beispiel verändert sich die Karte automatisch.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
```

```
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="leaflet-heat.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.1555 , 7.591838], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var points = [];
var points1 = [[50.1555 , 7.591838 ],...];
var points2 = [[50.1555 , 7.591838 ],...];
var points3 = [[50.1555 , 7.591838 ],...];
var heat = L.heatLayer(points).addTo(mymap);

x=1;
var name='';

var interval = setInterval(function(){run()},2000);

function run(){
mymap.removeLayer(heat);
name="points"+x.toString();
heat = L.heatLayer(
window[name],
{blur:15,maxZoom:10,radius:25,gradient:{0.4: 'blue', 0.65: 'lime',
1: 'red'}})
.addTo(mymap);
if (x == 3) {
x=1;
} else {
x++;
}
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

```
<!--index_958.html-->
```

Was passiert im Beispiel genau? Als Erstes integrieren wir wieder das notwendige Skript. Danach erstellen wir die vier Arrays und nennen diese `points`, `points1`, `points2` und `points3`. Jeder Array enthält unterschiedliche Daten. Mit dem ersten leeren Array erstellen wir als Nächstes die Schicht mit den Daten für die Heatmap, also das Objekt des Typs `Heatlayer`. Die Hilfsvariable `x` in Verbindung mit der Methode `setInterval()` ermöglicht es uns dann, die Heatmap Ebenen fortlaufend mit anderen Daten anzeigen.

In diesem Beispiel haben ich zwei JavaScript Elemente verwendet, die vielleicht erklärungsbedürftig sind. Als Erstes habe ich den Variablennamen mit `window[name]` zusammengesetzt. Was bedeutet dies genau? Da es sich bei Javascript um Objekte, wird jede Variable in einem globalen Objekt gespeichert. Wenn Sie also die Variable `points1` im globalen Bereich initialisieren – also nicht in einem Funktionskontext –, schreiben Sie diese Variablen implizit in ein globales Objekt. In einem Browser ist dies das Objekt `window`. Der Wert dieser Variablen kann mit der Punkt oder Klammer Notation abgerufen werden. Also entweder mit `var name = window.points1;` oder mit `var name = window['points1'];`

Außerdem haben wir die Methode `setInterval()` eingesetzt. Mit dieser Methode können Sie eine Funktion wiederholt aufrufen. Hierbei können Sie ein Intervall zwischen den einzelnen Aufrufen definieren.

So wie im nachfolgenden Bild sieht die Karte nur alle 6 Sekunden aus. In der Zwischenzeit wechselt die Ansicht zweimal.

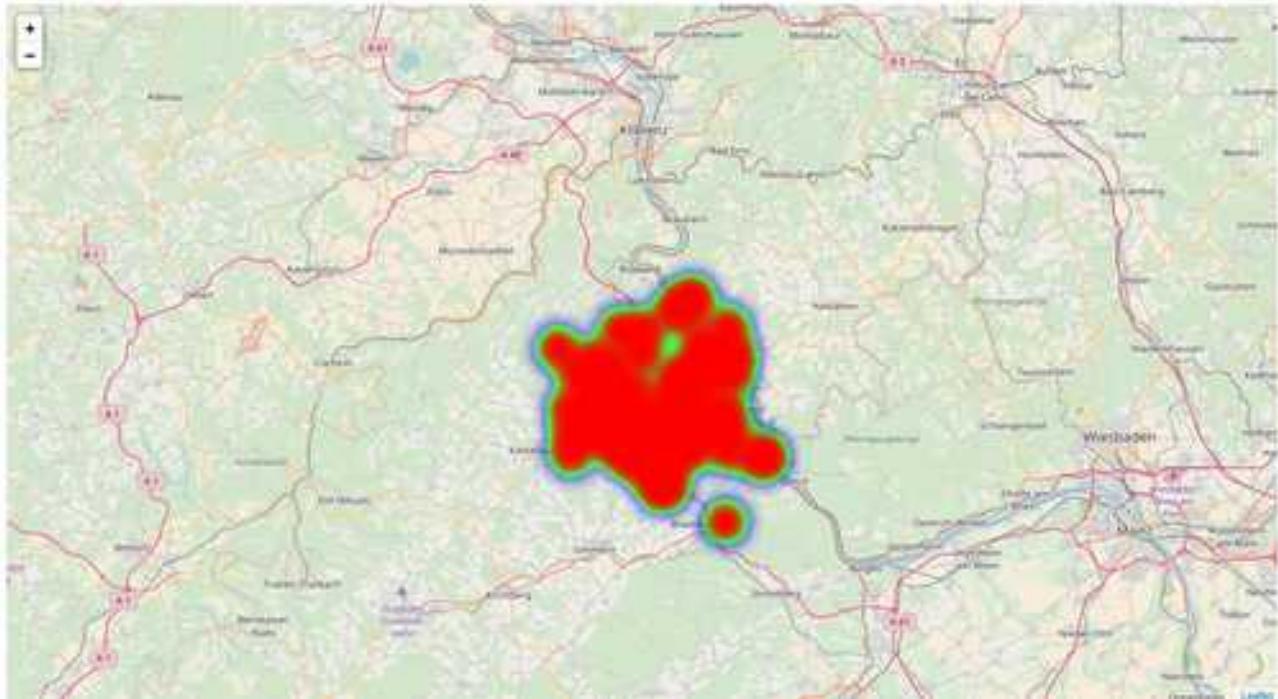


Abbildung 41
950.png

Choroplethenkarte

Im vorherigen Kapitel haben wir eine Heatmap zum visualisieren von Daten verwendet. Wir haben zunächst mithilfe des Skripts Leaflet.heat Bereiche, in denen Punkte dichter vorkommen, farblich hervorgehoben. Dann haben wir zusätzlich das Skript heat.js geladen und Bereiche, in denen die Punkte eine hohe Intensität haben, farblich besonders markiert. Eine [Choroplethenkarte](#) macht erst einmal nichts anderes. Sie visualisiert die Dichte oder die Intensität bestimmter Objekte.

Was genau ist eine Choroplethenkarte?

Im vorhergehenden Kapitel hatte ich Ihnen erklärt, dass eine Heatmap ein Raster über die Karte legt. Je nachdem wie die Punkte in diesem Raster liegen, werden Farben sichtbar. Eine Choroplethenkarte verwendet kein separates Raster. Sie kennzeichnet relativ zu einem Polygon. Ein Polygon kann zum Beispiel ein Land sein. In Kapitel *Die Karte mit Daten bestücken* hatte ich die Besonderheit eines Polygons beschrieben. Dieses Vieleck hat eine Grenzlinie die einen Innenbereich und einen Außenbereich voneinander abgrenzt. Eine populäre Choroplethenkarte, die Sie sicherlich schon einmal gesehen haben, ist die Darstellung der Bevölkerungsdichte eines Gebietes auf der Erde. Ich habe hier ein Beispiel erstellt, welches genau dies tut. Ich zeige, wie Sie eine Karte, die die Bevölkerungsverteilung in Rheinland-Pfalz grafisch darstellt, erstellen können.

Choroplethenkarten in Leaflet

Das Schöne ist, dass wir mit Leaflet keine zusätzlichen Plugins benötigen. Leaflet ist wie dafür gemacht, GeoJSON Daten als Choroplethenkarte anzuzeigen. Beginnen tun wir ganz vorne mit dem Klären der Frage: Wo bekommen Sie die Daten her?

Open Data

Wenn Sie nicht selbst über Daten verfügen, können sie auf jede Menge Open Data zugreifen.

[Open Data](#), also offene Daten, sind Daten die von jedem ohne jegliche Einschränkungen verwendet und weitergegeben werden dürfen. Warum gibt es Open Data? Viele Menschen vertreten die Meinung, dass frei nutzbare Daten zu mehr Transparenz und Zusammenarbeit führen. Die Bereitstellung offener Daten durch öffentliche Einrichtungen wird als eine Voraussetzung für [Open Government](#), also der Öffnung von Regierung und Verwaltung gegenüber der Bevölkerung und der Wirtschaft, angesehen. Die Befürworter von Open-Data teilen somit viele Argumente mit den Befürwortern von [Open-Source](#).

[GeoJSON Utilities](#) ist ein Projekt, welches den Export von Gemeindeflächen, Landkreisflächen und Bundeslandflächen in Deutschland im GeoJSON Format ermöglicht. Jede exportierte Fläche enthält zusätzliche Eigenschaften wie die Einwohnerzahl und Fläche in Quadratmeter. Ich habe diese Daten für Rheinland-Pfalz exportiert. Im nächsten Beispiel zeige ich Ihnen, wie Sie aus dieser GeoJSON-Datei schnell selbst Chorophletenkarten erstellt können.

[GeoJSON Utilities](#) ist übrigens auch eine mit Leaflet erstellte Anwendung.

Die GeoJSON-Datei, die ich heruntergeladen haben, ist relativ groß. Deshalb habe ich die Daten in einer separaten Datei abgelegt. Ich schlage Ihnen vor, dies auch zu tun. So bleibt Ihre HTML-Datei übersichtlich und Sie können sich voll und ganz auf das Erstellen der Karte konzentrieren. Wenn Sie die Daten in einer JavaScript Datei ablegen, können Sie diese gleichzeitig als Variabel deklarieren. Wie das geht, zeigt ihnen das nachfolgende Codebeispiel. Zunächst sehen Sie die Datei, die die GeoJSON Daten enthält. Dieser Datei habe ich den Namen `gemeinden.js` gegeben.

```
var ct =
```

```
{  
  "type": "FeatureCollection",  
  "crs": {  
    "type": "name",  
    "properties": {"name": "urn:ogc:def:crs:OGC:1.3:CRS84"}},  
  "source": "© GeoBasis-DE / BKG 2013 (Daten verändert)",  
  "features": [  
    {"type": "Feature",  
      "properties": {  
        "ADE": 6,  
        "GF": 4,  
        "BSG": 1,  
        "RS": "073345001001",  
        "AGS": "07334001", "SDV_RS": "073345001001",  
        "GEN": "Bellheim",  
        "BEZ": "Gemeinde", "IBZ": 64, "BEM": "gemeinschaftsangehörig", "NBD": "ja",  
        "SN_L": "07", "SN_R": "3", "SN_K": "34", "SN_V1": "50", "SN_V2": "01", "SN_G": "001",  
        "FK_S3": "K", "NUTS": "DEB3E", "RS_0": "073345001001", "AGS_0": "07334001",  
        "WSK": "2009/01/01", "DEBKID": "DEBKDL20000E5ZS",  
        "destatis": {  
          "RS": "073345001001", "area": 20.44, "area_date": "31.12.2014", "population": 8519, "population_m": 4198, "population_w": 4321, "population_density": 417, "zip": "76756", "center_lon": "8.284042", "center_lat": "49.190542", "travel_key": "J16", "travel_desc": "Pfalz", "density_key": "02", "density_desc": "mittlere Besiedlungsdichte"},  
        "wikipedia": {  
          "cid": "http://www.wikidata.org/entity/Q552382", "name": "Bellheim",  
          "AGS": "07334001", "_official_website": "http://www.bellheim.de/", "_local_dialing_code": "07272", "_postal_code": "76756"}},  
        "geometry": { "type": "Polygon", "coordinates": [[[[8.276302148832034, 49.21145214735215],  
          [8.295112386392997, 49.21205183793759],  
          [8.276302148832034, 49.21145214735215]]]]}}  
      }  
    }  
  ]  
}
```

```
...
[8.259740044810764, 49.21103338616947],
[8.276302148832034, 49.21145214735215]]} },
{"type": "Feature",
"properties":
{"ADE": 6, "GF": 4, "BSG": 1, "RS": "073405003205", "AGS": "07340205", "SDV_RS": "073405003205", "GEN": "Bottenbach", "BEZ": "Gemeinde", "IBZ": 64
...
}
```

In den Properties der einzelnen GeoJSON Features sind unter anderem der Regionalschlüssel (RS), der geographische Name (GEN), die amtliche Bezeichnung der Verwaltungseinheit (BEZ) und die Destatis-Daten – insbesondere Fläche in Quadratmetern und Einwohnerzahlen – enthalten

Das Beispiel zeigt nur einen Ausschnitt der Datei `gemeinden.js`. Beachten Sie, dass es sich nicht um reines GeoJSON handelt. Ganz genau handelt es sich um JavaScript. Dieser JavaScript Code deklariert eine Variable, nämlich die Variable `ct`. Einbinden können Sie den JavaScript-Ausschnitt genau wie jedes andere Skript in Ihre HTML-Datei mit der Zeile

```
<script src="gemeinden.js"></script>
```

Wenn die Datei richtig eingebunden ist, können Sie auf die Daten in der Datei ab nun in der Variablen `ct` zugreifen.

Farben

Der nächste Schritt zum Fertigstellen der Choroplethenkarte ist die Farbauswahl. Beim Erstellen der Heatmaps im vorhergehenden Kapitel haben wir nur Farben übergeben und das jeweilige Plugin hat diese entsprechend zugewiesen. Beim Erstellen der Choroplethenkarte nutzen wir kein Plugin. Deshalb müssen wir uns selbst um diese Aufgabe kümmern. Deshalb definieren wir eine Funktion, die je nach Wert eine bestimmte Farbe errechnet und zurück gibt. Idealerweise kennen Sie den höchsten Wert und den niedrigsten Wert in der möglichen Datenmenge. Nun können Sie, je nach Verteilung der Werte bestimmten Wertbereichen eine Farbe zuordnen.

Der nachfolgende Programmcodeausschnitt nimmt einen Parameter entgegen und berechnet anhand des Wertes dieses Parameters eine Farbe. Die Formel in

der Funktion ist so definiert, dass höhere Werte dunklere Farben und niedrigere Werte hellere Farben als Ergebnis ausgeben.

```
function color(x) {  
  return x > 1000 ? '#990000' :  
    x > 750 ? '#d7301f' :  
    x > 500 ? '#ef6548' :  
    x > 250 ? '#fc8d59' :  
    x > 200 ? '#fdbb84' :  
    x > 100 ? '#fdd49e' :  
    x > 0 ? '#fee8c8' : '#fff7ec';  
}
```

Stile

Als Nächstes benötigen wir eine Funktion, die das Zuweisen des passenden CSS Stylesheets zu den Daten übernimmt. Nur so können wir jedes Feature individuell ansehen und ihm den passenden Stil zuweisen. Im nachfolgenden Programmcodeausschnitt sehen Sie eine Funktion, die als Parameter ein Feature erwartet. Je nach Eigenschaft im Feature gibt die Funktion die passenden Optionen zurück. Sie erkennen sicherlich sofort, dass die wesentliche Option die mit dem Namen `fillColor` ist. Hier ist genau die Stelle, an der wir die eben erstellte Funktion `color()` aufrufen und dieser als Parameter den Wert `feature.properties.destatis.population`, also die Bevölkerungsanzahl, mitgeben.

```
function myStyle(feature) {  
  return {  
    fillColor: color(feature.properties.destatis.population),  
    weight: 1,  
    opacity: 1,  
    color: 'white',  
    fillOpacity: 0.85  
  };  
}
```

Das vollständige Beispiel

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="gemeinden.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([49.9555 , 7.591838], 8);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

function color(x) {
return x > 1000 ? '#990000' :
x > 750 ? '#d7301f' :
x > 500 ? '#ef6548' :
x > 250 ? '#fc8d59' :
x > 200 ? '#fdbb84' :
x > 100 ? '#fdd49e' :
x > 0 ? '#fee8c8' : '#fff7ec';
}

function myStyle(feature) {
return {
fillColor: color(feature.properties.destatis.population),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}
}
```

```
var geoJsonLayer = L.geoJson(ct, {style: myStyle}).addTo(mymap);  
</script>  
</body>  
</html>  
<!--index_957.html-->
```

In Ihrem Browser sollte die Karte nun so aussehen, wie in der nächsten Abbildung dargestellt ist.

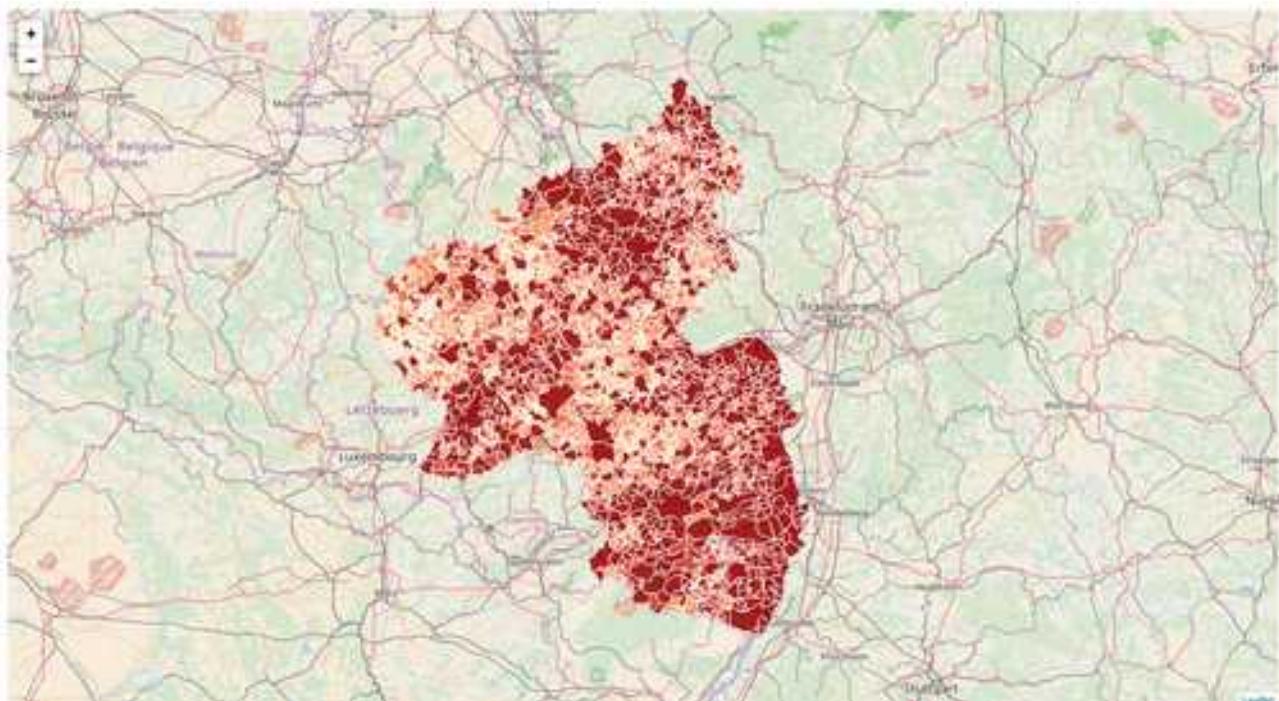


Abbildung 42

947.png

In diesem Beispiel haben wir einen festen Wert ganz unabhängig von anderen Werten verwendet. Wir haben die Bevölkerungszahl unabhängig von der Flächengröße der Gemeinde als Wert für die Farbe verwendet. In dieser Karte können wir zwar ablesen, wo die Gemeinden mit hoher Bevölkerungszahl sich befinden und wo die Gemeinden mit niedriger Bevölkerungsanzahl sind. Wie dicht die Besiedelung ist, sagt diese Karte aber noch nicht aus. Hierzu müssten wir die Bevölkerungszahl noch relativ zur Fläche der Gemeinde setzen. Dies tun wir nun im nächsten Beispiel.

Normalisierte Choroplethenkarten

Es kommt sicherlich sehr oft vor, dass Sie mit einer Choroplethenkarten nicht nur einen absoluten Wert darstellen möchten. Oft möchten Sie die Daten zu anderen Daten ins Verhältnis stellen. Zum Beispiel könnten Sie den Anteil von Menschen die älter als 60 Jahre sind darstellen wollen. Da Geodaten einen

Bezug zu einer Fläche haben, wird es meist so sein, dass Sie die Daten in Relation zu dieser Fläche visualisieren möchten – diese also normalisieren möchten.

In unser GeoJSON Datei ist die Fläche der Gemeinde in der Eigenschaft `area` gespeichert. Wir können es uns also einfach machen und Zahl dir für die Fläche einer Gemeinde gespeichert ist, zugreifen. Um die Choroplethenkarte, die wir im vorhergehenden Kapitel erstellt haben, zu normalisieren ist somit nur ein Schritt – nämlich das Teilen der Bevölkerungsanzahl durch die Fläche des betreffenden Bereichs – erforderlich. Sehen wir uns aber das ganze Beispiel noch einmal zusammenhängend an.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="gemeinden.js"></script>
</head>
<body>
<button onclick="total()">Population</button>
<button onclick="density()">Population/Fläche</button>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([49.9555 , 7.591838], 8);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
function color(x) {
return x > 1000 ? '#990000' :
x > 750 ? '#d7301f' :
x > 500 ? '#ef6548' :
x > 250 ? '#fc8d59' :
x > 200 ? '#fdbb84' :
x > 100 ? '#fdd49e' :
```

```
x > 0    ? '#fee8c8' : '#fff7ec';
}

function densityColor(x) {
return x > 500 ? '#990000' :
x > 400 ? '#d7301f' :
x > 300 ? '#ef6548' :
x > 200 ? '#fc8d59' :
x > 100 ? '#fdbb84' :
x > 50  ? '#fdd49e' :
x > 0   ? '#fee8c8' : '#fff7ec';
}

function myStyle(feature) {
return {
fillColor: color(feature.properties.destatis.population),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}

function myDensityStyle(feature) {
return {
fillColor:
densityColor(feature.properties.destatis.population/feature.properties.destatis.area),
weight: 1,
opacity: 1,
color: 'white',
fillOpacity: 0.85
};
}

function total() {
var geoJsonLayer = L.geoJson(ct, {style: myStyle}).addTo(mymap);
removeLayer(densitylayer);
}

function density() {
```

```

var densitylayer=L.geoJson(ct, {style:
myDensityStyle}).addTo(mymap);
removeLayer(geoJsonLayer);
}
</script>
</body>
</html>
<!--index_956.html-->

```

Das vorhergehende Codebeispiel baut dem davor erstellten Codebeispiel auf. Neben der `style` Methode fügen wie eine weitere Methode ein. Die Methode `densityStyle` gibt, wie der Name schon sagt, die Farbe für die Relative angabe zurück. Damit wir die beiden Choroplethenkarte Ebenen vergleichen können, habe ich eine Schaltfläche in das HTML Dokument integriert. Je nachdem welche Schaltfläche angeklickt wird, wird die passende Formel ausgeführt. Einmal die, die absolute Werte anzeigt, ein anderes mal die, die die relativen Werte ausgibt.

Nun werden Sie feststellen, dass viele flächenmäßig große Bereiche in ländlichen Gegenden, nicht mehr dunkel eingefärbt sind. Im ersten Beispiel, in dem wir mit den absoluten Zahlen gearbeitet haben, waren diese Bereiche teilweise dunkel. Die Bevölkerungszahl ist aber in Relation zur Fläche doch nicht so hoch.

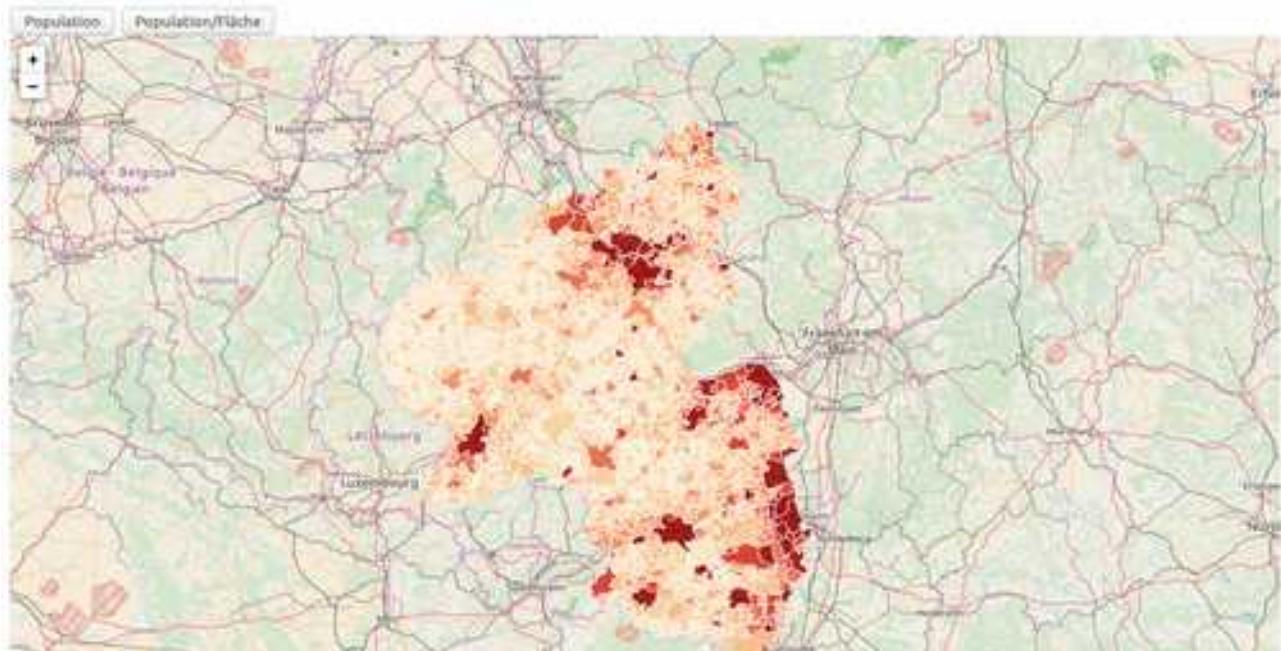


Abbildung 43
948.png

In diesem Kapitel haben wir ...

In diesem Kapitel haben Sie alles das, was Sie in den ersten Teilen kennen gelernt haben, zum visualisieren eingesetzt. Sie haben Plugins kennen gelernt, die Sie beim Erstellen von Heatmaps unterstützen. Sie können nun eine Heatmap erstellen, mit der interagiert werden kann und auch eine die Daten animiert anzeigt. Sie kennen die Besonderheiten von Choroplethenkarten und wissen, wann absolute und wann relative Werte sinnvoll sind und wie Sie die Anzeige von absoluten und relativen Werten selbst umsetzen können.

Im nächsten Kapitel werden wir noch einmal den Schwerpunkt auf die individuelle Gestaltung setzen. Hier geht es um benutzerdefinierte Marker.

Custom Markers

Sie wissen nun wie Sie die unterschiedlichsten Geodaten auf Ihrer Karte anzeigen können und auch wie Sie mit den Geodaten ein Thema visualisieren können.

In diesem Kapitel werden wir ...

In diesem Kapitel werden wir nun über die reine Anzeige hinaus noch einen Schritt weiter gehen. Nun geht es darum, der Karte eine individuelle Note zu geben. Wir sehen uns an, wie Sie Marker beliebig gestalten können. Außerdem sehen wir uns Plugins an, die Ihnen diese Arbeit erleichtert.

Ein individueller Marker auf Ihrer Karte

Wenn Leaflet einen Marker auf einer Karte anzeigt, werden gleich zwei Bilddateien angezeigt. Zunächst wird das eigentliche Bild an die passende Stelle auf die Karte gelegt. Danach wird ein Schatten zu diesem Bild hinzugefügt. Mit einem passenden Schatten fallend die Marker eher ins Auge. Der Schatten verleiht einem Marker eine Tiefe. So hebt dieser sich besser von der Karte ab.

Wenn Sie Leaflet auf Ihren Rechner kopiert haben, sehen Sie unter den kopierten Daten – sofort in der ersten Ebene – einen Unterordner, der `images` heißt. Dieser Ordner enthält die Imagedateien, die angezeigt werden, wenn kein individuelles Image angegeben ist. Ich habe die Bilder hier nachfolgend abgedruckt. Wenn Sie dieses Buch bisher durchgearbeitet haben, kommen Ihnen die Bilder sicher bekannt vor.



Abbildung 44

978.png

Oft ist es so, dass das Bekannte vertraut ist und man sich deshalb damit sicher und wohl fühlt. Manchmal möchte man aber aus der Reihen tanzen. Wenn Sie auf Ihrer Karte außergewöhnliche Stellen mit einem Marker markieren möchten, dann soll dieser besondere Marker sicher aus der Reihe tanzen, oder? Wenn Sie an dieser Stelle keine blaue Einheitsgrafik anzeigen möchten, dann können Sie auch Ihre eigene Grafik anzeigen. Haben Sie schon eine schöne Grafik für Ihren Marker und den passenden Schatten dazu? Wie Sie ein eigenes Image mit einem Grafikprogramm erstellen, gehört nicht zum Thema dieses Buches. Hier möchte ich Ihnen nur ein paar Punkte aufzählen, die Sie beachten sollten. Falls Sie keine Grafiken besitzen und auch nicht selbst Hand anlegen möchten, dann könne Sie entweder die Übungen wie ich mit den Beispielbildern im Leaflet Tutorial durcharbeiten – oder Sie blättern direkt weiter zum nächsten Kapitel. Dieses Kapitel bietet Ihnen einen Kompromiss. Sie benötigen keine eigenen Grafiken, können einem Marker aber trotzdem ein anderes Aussehen verleihen.

Sie möchten gerne selbst die Grafiken erstellen wissen aber noch nicht genau wie und womit? Dann sehen Sie sich doch einmal [GIMP](#) an. GIMP (GNU Image Manipulation Program) ist eine gute kostenlose Alternative zu [Photoshop von Adobe](#) und kommt mit zahlreichen professionellen Bearbeitungsfunktionen.

Wenn Sie zwei Bilder haben – also ein Bild, das Ihren Marker selbst darstellt und eines, das den Schatten zeigt – dann können wir diese als Marker in Ihre Karte einbinden. Ich habe hier zum Ausprobieren die Bild-Dateien aus dem Leaflet Tutorial [Markers with Custom Icons](#) verwendet.

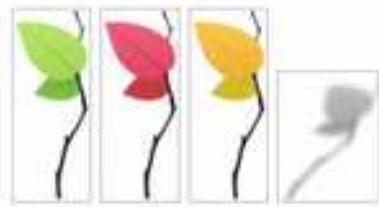


Abbildung 45

945.png

Im nachfolgenden Beispiel habe ich den Text, der für die Anzeige des benutzerdefinierten Markers verantwortlich ist, fett formatiert.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 10);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var greenIcon = L.icon({
iconUrl: 'http://leafletjs.com/examples/custom-icons/leaf-
green.png',
shadowUrl: 'http://leafletjs.com/examples/custom-icons/leaf-
shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94],
```

```

shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
});

L.marker([50.27264, 7.26469], {icon:
greenIcon}).addTo(mymap).bindPopup("Ich bin ein Marker mit einem
individuellen Image.");
</script>
</body>
</html>
<!--index_955.html-->

```

Sehen wir uns diese Codeteile einmal genau an. Zunächst einmal sticht die Instanziierung des Bildes hervor.

```

var greenIcon = L.icon({
iconUrl: 'http://leafletjs.com/examples/custom-icons/leaf-green.png',
shadowUrl: 'http://leafletjs.com/examples/custom-icons/leaf-shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94],
shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
});

```

Die Bedeutung der einzelnen Optionen habe ich in der nachfolgenden Abbildung veranschaulicht. Die Schwierigkeit liegt darin, die Position des Icons mit der Stelle auf der Erde, die markiert werden soll, zu harmonisieren. Das Bild ist in der Regel größer als der Punkt, der mit ihm beschrieben werden soll. So können Sie mit der Option `iconSize: [38, 95]` die Größe des Bildes mitgeben. Mit der Option `shadowSize: [50, 64]` können Sie die Größe des Schattens beeinflussen. Wenn Sie die Werte nicht setzen, wird die Originalgröße verwendet. Die Option `iconAnchor: [22, 94]` gibt Ihnen nun die Möglichkeit, die Stelle an der das Bild in die Karte eingefügt werden soll, zu definieren. Ohne ein Setzen dieser Option, würde die linke obere Ecke des Bildes an der Stelle, die markiert werden soll, beginnen. In der Regel ist es aber so, dass man das Bild mittig oder vielleicht sogar über dieser Stelle einfügen möchte. `IconAnchor: [22, 94]` fügt das Bild 22 Pixel weiter links

und 94 Pixel oberhalb der zu markierenden Stelle ein. Wenn Sie sich die nachfolgenden Abbildung ansehen, wird dies klar. Der rote Punkt stellt in der Abbildung die zu markierende Stelle dar. Für die Option `shadowAnchor: [4, 62]` gilt das gleiche wie für `iconAnchor`. Die Belegung der Wert in der Option `popupAnchor: [-3, -76]` ist meiner Meinung nach etwas verwirrend. Hier müssen Sie ein Minuszeichen voran stellen, wenn Sie das Pop-up Fenster nach links oben, relativ zur zu markierenden Position, verschoben möchten.

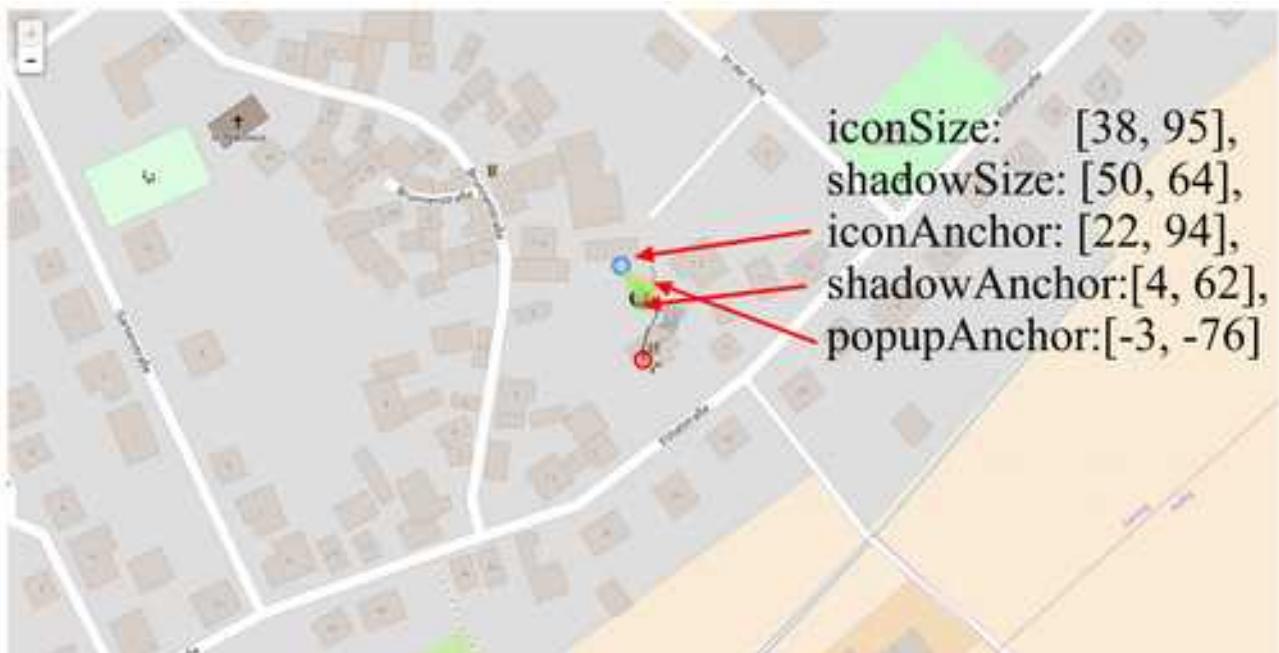


Abbildung 46

Im Kapitel *Die Karte mit Daten bestücken* genau im Unterkapitel genau im Unterkapitel *Punkte und Marker* haben wir schon Marker mit Optionen angelegt. Im nächsten Codeschnipsel sehen Sie nun, dass Sie auch das Bild zum Marker – also das Icon – als Option angeben können. Im Programmcode geben Sie dazu einfach den Namen des eben erstellen `L.Icon` Objektes an.

```
L.marker([50.27264, 7.26469], {icon:  
greenIcon}).addTo(mymap).bindPopup("Ich bin ein Marker mit einem  
individuellen Image");
```

Wenn Sie das vorhergehenden Programmcode angezeigte HTML Dokument öffnen, sehen Sie eine Karte auf der das grüne Icon als Marker an der festgelegten Koordinate – passend positioniert – erscheint.

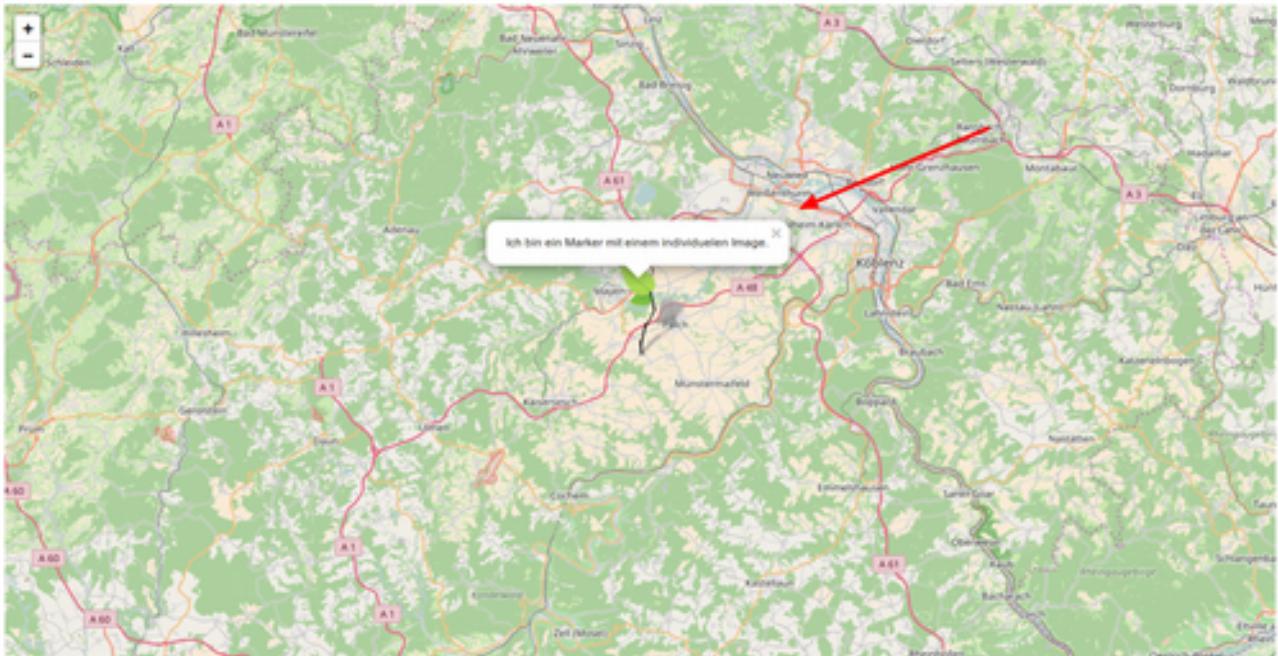


Abbildung 47

977.png

Eigenschaften eines individuellen Marker

Einen eigenes Marker Bild erstellen Sie in Leaflet mithilfe der Kasse [L.icon](#).

Das haben Sie eben praktisch gesehen. Diese Klasse bietet Ihnen zehn Optionen.

- **iconUrl:**
Die `iconUrl` müssen Sie auf alle Fälle angeben. Die `IconUrl` muss die Adresse zur Bilddatei enthalten – absolut oder relativ zu Ihrem Skriptpfad.
- **iconRetinaUrl:**
Die `iconRetinaUrl` bietet Ihnen optional die Möglichkeit die Adresse einer für Retina Bildschirme optimierten Version des Bildes – absolut oder relativ zum Skriptpfad – anzugeben.
- **iconSize:**
Die `iconSize` beeinflusst die Größe in der das Bild angezeigt wird.
- **IconAnchor:**
Die Pixel Koordinaten an der das Bild eingefügt werden soll – relativ zu oberen linken Ecke.
- **PopupAnchor:**
Die Pixel Koordinaten des Punktes, von dem Popups "öffnen", relativ zum Symbol Anker.

- `ShadowUrl`:
Diese Option enthält die Adresse zur Imagedatei, die den Schatten darstellen soll. Wenn nichts angegeben ist, wird kein Schattenbild erstellt.
- `ShadowRetinaUrl`:
Mit dieser Option können Sie ein Bild angeben, dass speziell für Retina Displays optimiert wurde.
- `shadowSize`:
Die Höhe und Breite des Bildes, dass den Schatten darstellen soll in Pixeln.
- `shadowAnchor`:
Die Pixel Koordinaten an der das Schattenbild eingefügt werden soll, können Sie mit `ShadowAnchor` übergeben. Ansonsten gilt für diese Option das Gleiche, was ich bei der Option `iconAnchor` geschrieben habe.
- `className`:
Mit der Option `className` können Sie den Namen einer CSS-Klasse, die beiden Bildern, also dem Schattenbild und dem eigentlichen Marker Bild hinzugefügt wird, definieren.

Exkurs:

Die neuen, hochauflösenden Displays haben eine höhere Pixeldichte als gewöhnliche Monitore. Auf der gleichen Fläche werden etwa viermal so viele Pixel dargestellt. Der Vorteil dieser Technologie liegt darin, dass die Pixel nun so klein sind, dass das menschliche Auge sie nicht mehr auflösen kann. Das Ergebnis sind sehr scharfe Grafiken und Texte. Damit das Bild nun auf einem HiDPI (High Dots Per Inch) Bildschirm, also einem Retina Display, scharf dargestellt wird, muss es mit mindestens 2-facher Breite und Höhe zur Verfügung gestellt werden. Eine Pixelgrafik die bei gewöhnlicher Auflösung das ganze Display ausfüllt, würde eigentlich auf einem Retina Display der gleichen Größe nur ein Viertel des Displays einnehmen. Ein Pixel der Grafik entspricht ja auch auf dem Retina-Display einem Pixel, nur das viermal so viele und dafür kleinere Pixel abgebildet werden. Damit die Pixelgrafiken nicht plötzlich alle zu klein dargestellt werden, rechnen die Geräte die Grafiken selbstständig um. Dadurch geht allerdings Qualität verloren. Pixelgrafiken sehen auf dem Retina-Display daher unscharf aus. Um dieses Problem zu umgehen, prüft Leaflet die Auflösung des Anzeigegerätes. Anschließend werden die Marker Bilder – sofern vorhanden – in hoher Auflösung angezeigt.

Die Klasse L.Icon erweitern

So, nun haben Sie einen benutzerdefinierten Marker erstellt und möchten weitere Marker kreieren. Schön, dass Leaflet objektorientiertes Arbeiten unterstützt. So können Sie sich das Erstellen von neuen Marker Objekten sehr einfach machen. Erweitern Sie einfach die Klasse L.Icon und geben alle gemeinsamen Eigenschaften einmal an. Nur die besonderen Eigenschaften des Markers müssen Sie separat angeben. Das nachfolgende Codebeispiel zeigt Ihnen, wie Sie drei unterschiedliche Marker, die aber gemeinsame Eigenschaften haben, mithilfe von einer Eltern-Klasse erstellen können. So können die drei Marker von dem Elternteil die gemeinsamen Eigenschaften erben.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 9);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var LeafIcon = L.Icon.extend({
options: {
shadowUrl: 'leaf-shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94],
shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
}

```

```

});
```

```

var greenIcon = new LeafIcon({iconUrl:
'http://leafletjs.com/examples/custom-icons/leaf-green.png'});
var redIcon = new LeafIcon({iconUrl:
'http://leafletjs.com/examples/custom-icons/leaf-red.png'});
var orangeIcon = new LeafIcon({iconUrl:
'http://leafletjs.com/examples/custom-icons/leaf-orange.png'});
L.marker([50.27264, 7.26469], {icon:
greenIcon}).addTo(mymap).bindPopup("Ich bin ein grüner Marker.");
L.marker([50.27264, 6.26469], {icon:
redIcon}).addTo(mymap).bindPopup("Ich bin ein roter Marker.");
L.marker([50.27264, 8.26469], {icon:
orangeIcon}).addTo(mymap).bindPopup("Ich bin ein oranger
Marker.");
</script>
</body>
</html>
<!--index_954.html-->
```

Dieser Programmcodeabschnitt hat Ähnlichkeit mit dem im vorherigen Beispiel. Wenn Sie genau hinsehen, finden Sie aber Unterschiede. Wir erstellen im ersten Schritt kein neues `Icon` Objekt, sondern erweitern die Klasse `L:Icon`. Mit der Klasse `LeafIcon`. Wir packen die Optionen in ein Objekt und legen nur die gemeinsamen Optionen fest. Die URL des Bildes ist die Option, die von Marker zu Marker unterschiedlich ist. Deshalb geben wir diese beim Erweitern der Klasse in den Optionen noch nicht an.

```

var LeafIcon = L.Icon.extend({
options: {
shadowUrl: 'leaf-shadow.png',
iconSize: [38, 95],
shadowSize: [50, 64],
iconAnchor: [22, 94],
shadowAnchor: [4, 62],
popupAnchor: [-3, -76]
}
});
```

Im zweiten Schritt erstellen wir drei Instanzen des LeafIcon, also des erweiterten L.Icon

```
var greenIcon = new LeafIcon({iconUrl:  
'http://leafletjs.com/examples/custom-icons/leaf-green.png'});  
var redIcon = new LeafIcon({iconUrl:  
'http://leafletjs.com/examples/custom-icons/leaf-red.png'});  
var orangeIcon = new LeafIcon({iconUrl:  
'http://leafletjs.com/examples/custom-icons/leaf-orange.png'});
```

Und zu guter Letzt fügen wir die Marker mit dem zugehörigen Icon an die passende Stelle auf der Karte zum Kartenobjekt hinzu.

```
L.marker([50.27264, 7.26469], {icon:  
greenIcon}).addTo(mymap).bindPopup("Ich bin ein grüner Marker.");  
L.marker([50.27264, 6.26469], {icon:  
redIcon}).addTo(mymap).bindPopup("Ich bin ein roter Marker.");  
L.marker([50.27264, 8.26469], {icon:  
orangeIcon}).addTo(mymap).bindPopup("Ich bin ein oranger  
Marker.");
```

Exkurs:

Sie haben vielleicht bemerkt, dass wir das Schlüsselwort `new` für die Erstellung von LeafIcon Instanzen verwendet haben. Warum haben wir vorher alle Leaflet-Klassen ohne `new` erstellt? Die Antwort ist einfach: Die echten Leaflet Klassen sind mit einem Großbuchstaben – beispielsweise `L.Icon` – benannt und diese müssen mit `new` erstellt werden. Es gibt aber Shortcuts mit Kleinbuchstaben – `L.icon` – die aus Bequemlichkeitsgründen von den Leaflet-Programmierern für Sie erstellt wurden:

```
L.icon = function icon(options) {  
return new L.Icon(options);  
};
```

Die Funktion `L.icon` können Sie sich auf Github in der Datei [icon.js](#) ansehen.

Leaflet setzt hier das Entwurfsmuster [Fabrikmethode](#) (englisch factory method) ein. Das Muster beschreibt, wie ein Objekt durch Aufruf einer Methode anstatt durch direkten Aufruf eines Konstruktors erzeugt wird.

Im nachfolgenden Bild sehen Sie das Ergebnis. Jeder Marker wird nun mit einem individuellen Icon erstellt. Die meisten Optionen sind gleich – allerdings hat jeder Marker seine eigene Farbe.

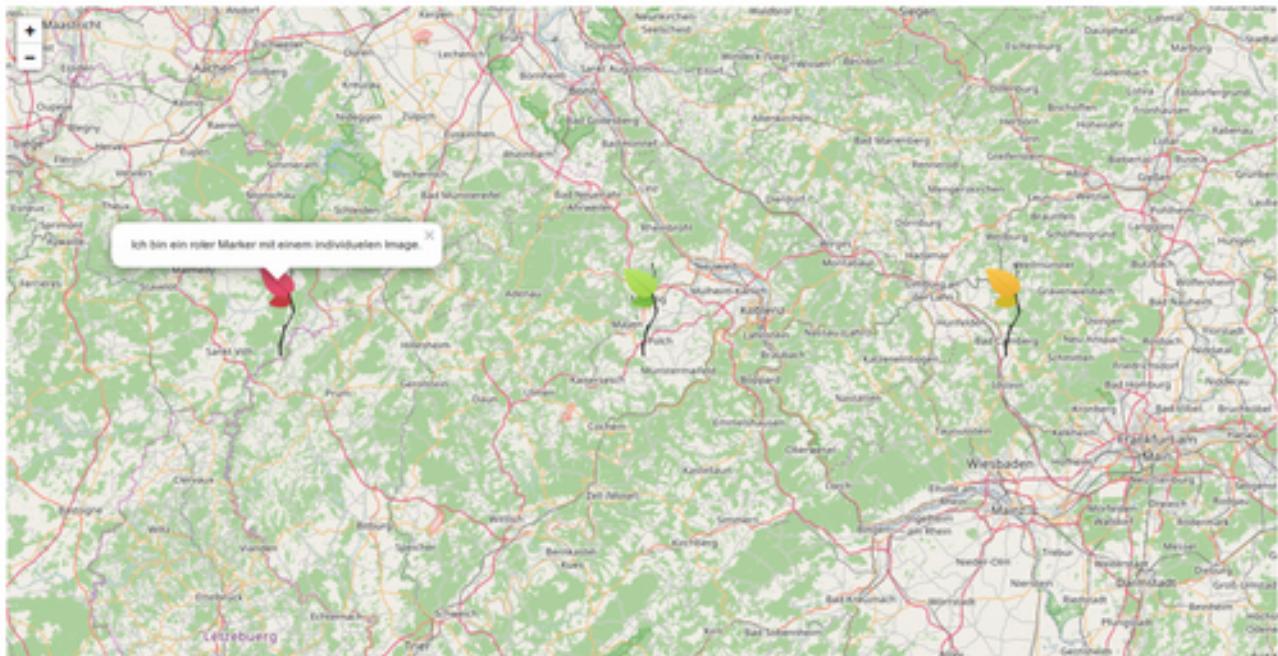


Abbildung 48
976.png

Ein Marker Plugin

Sie wissen nun, wie Sie einen Marker mit einem standardisierten Aussehen einfügen und können einen Marker mit einem eigenen Image belegen. Leider haben Sie aber kein eigenes Image und haben auch nicht viel Erfahrung mit einem Grafikprogramm. Sie können kein professionell aussehendes individuelles Image hervor zaubern oder haben einfach nicht die Zeit dazu. Trotzdem möchten Sie Ihrer Karte ein besonderes Aussehen verleihen. In diesem Fall bietet Ihnen dieses Kapitel eine Lösung. Ich stelle Ihnen zwei Plugins vor, die Sie bei der Erstellung von individuellen Marker Objekten unterstützen.

BeautifyIcon

[Leaflet.BeautifyIcon](#), ist ein einfaches Plugin, das bunte Marker ganz ohne eigene Grafik zu Leaflet hinzufügt. Trotzdem behalten Sie die volle Kontrolle über den Marker Stil. Konkret heißt das: Sie können über unbegrenzte Farben und viele Eigenschaften verfügen. Das Plugin Leaflet.BeautifyIcon bietet auch die Möglichkeit, Schriftart und Glyphen, also die grafische Darstellung von Schriftzeichen, anzupassen.



Abbildung 49

974.png

Damit Sie sich dieses besser vorstellen können, habe ich ein Beispiel erstellt.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css">
<link rel="stylesheet"
href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<link rel="stylesheet" href="leaflet-beautify-marker-icon.css">
<script src="leaflet-beautify-marker-icon.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
```

```
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 8);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

L.marker([50.27264, 7.26469], {
icon: L.BeautifyIcon.icon(
{iconSize: [50, 50]}
),
draggable: true
}).addTo(mymap).bindPopup("Ich bin ein beautify Marker");
options = {
icon: 'spinner',
spin: 'true',
borderColor: '#8A90B4',
textColor: 'white',
backgroundColor: '#8A90B4'
};

L.marker([50.27264, 6.26469], {
icon: L.BeautifyIcon.icon(options),
draggable: true
}).addTo(mymap).bindPopup("Ich bin ein beautify Marker");
options = {
icon: 'plane',
iconShape: 'marker',
borderColor: '#8D208B',
textColor: '#8D208B',
backgroundColor: 'transparent'
};

L.marker([50.27264, 8.26469], {
icon: L.BeautifyIcon.icon(options),
draggable: true
}).addTo(mymap).bindPopup("Ich bin ein beautify Marker");
</script>
</body>
</html>
<!--index_953.html-->
```

Was müssen Sie tun, wenn Sie das Plugin Leaflet.BeautifyIcon verwenden möchten? Zunächst einmal müssen Sie die notwendigen Skripts und Stylesheet Dateien einbinden. Das ist zum einen das Skript und die CSS-Datei zum Plugin selbst. Zum anderen können Sie über Leaflet.BeautifyIcon Drittdienste nutzen. Sie können Font Awesome CSS und Bootstrap CSS einbinden. Ich habe im Beispiel die CSS-Dateien von Font Awesome und von Bootstrap eingebunden, um Ihnen dies zu demonstrieren. Für dieses Beispiel wäre nur die CSS-Datei von Font Awesome CSS notwendig gewesen.

Mehr müssen Sie nicht tun. Sie können sofort einen Marker mit der Option icon: L.BeautifyIcon.icon(options) kreieren.

```
L.marker([50.27264, 8.26469], {  
  icon: L.BeautifyIcon.icon(options),  
  draggable: true  
}).addTo(mymap).bindPopup("Ich bin ein beautify Marker");
```

Sehen Sie sich die Optionen des Plugin Leaflet.BeautifyIcon an. Es macht Spaß diese zu erkunden. Wie die Optionen wirken, die ich verwendet haben, können Sie sich im nächsten Bild teilweise ansehen. Da sich eines der Icons dreht, erkennen Sie nur, wenn Sie die Datei selbst im Browser öffnen.

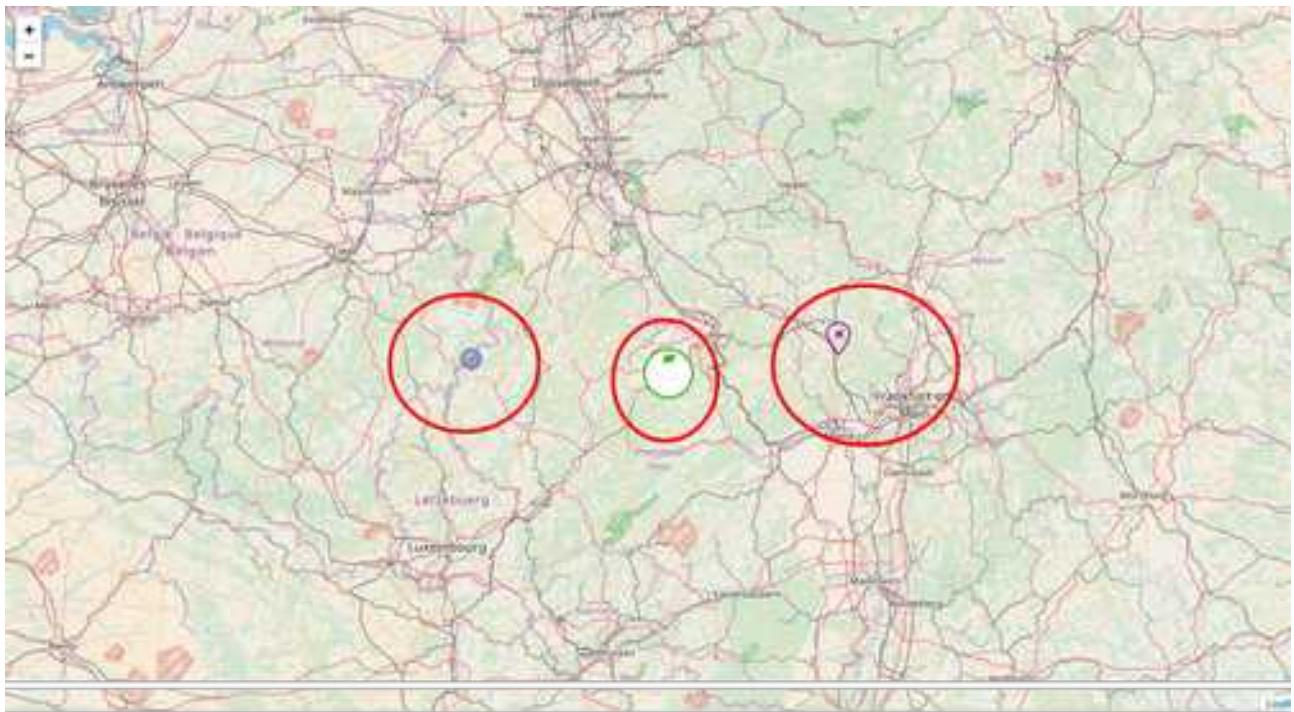


Abbildung 50
975.png

Ich hatte Ihnen ja schon geschrieben, dass es jede Menge Plugins für Leaflet gibt und dies gilt für den Bereich Marker besonders. Die meisten sind auf der Website von Leaflet aufgelistet. Diese Liste finden Sie unter der Adresse <http://leafletjs.com/plugins.html#markers-renderers>.

Cluster

Je nachdem welche Informationen Sie mit Ihrer Karte weitergeben möchten, kann es vorkommen, dass Sie sehr viele Marker benötigen. Wenn Sie mit vielen Marker Objekten arbeiten, sollten Sie beachten, dass diese das Laden der Karte verlangsamen. Außerdem kann es vorkommen, dass Marker nahe nebeneinander liegen und sich beim Zoomen überschneiden. Dies ist nicht benutzerfreundlich. Schön wäre es doch, wenn bei einer vergrößerten Ansicht alle Marker zu sehen sind – diese aber beim Hineinzoomen in die Karte zu Clustern zusammengefasst werden. So hat der Benutzer alle Informationen passend zur Kartenanzeige.

In diesem Kapitel erfahren Sie, wie Sie das Plugin [Leaflet.markercluster](#) zum Clustern von Marker Objekten verwenden und so eine große Anzahl von Marker Objekten auf einer Karte benutzerfreundlich und übersichtlich darstellen können. Sehen Sie sich das nachfolgende Beispiel an, um zu verstehen, wie das Clustern von Marker Objekten funktioniert.

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<link rel="stylesheet" href="MarkerCluster.css"/>
<link rel="stylesheet" href="MarkerCluster.Default.css"/>
<script src="leaflet.markercluster-src.js"></script>
<script src="points.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.219264, 7.19469], 13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);

var markers = L.markerClusterGroup(
//{showCoverageOnHover : false}
);

for (var i = 0; i < points.length; i++) {
var a = points[i];
var title = a[2];
var marker = L.marker(new L.LatLng(a[0], a[1]), { title: title });
marker.bindPopup(title);
markers.addLayer(marker);
}
mymap.addLayer(markers);
</script>
</body>
</html>
<!--index_952.html-->
```

todo Points anzeigen.

Das haben wir gemacht

So wie in der nachfolgenden Abbildung zu sehen ist könnte Ihre Karte aussehen.



Abbildung 51

973.png

Optionen, Methoden und Ereignisse

Todo

Optionen

Die folgenden Optionen sind standardmäßig aktiviert.

Standardmäßig aktiviert (boolesche Optionen)

- ShowCoverageOnHover:
Wenn Sie die Maus über einen Cluster bewegen, blendet sich ein Polygon ein, dass die Grenzen des Bereichs in dem die Marker sich befinden, anzeigt.
- ZoomToBoundsOnClick:
Wenn du auf einen Cluster klickst, zoomen wir auf seine Grenzen.
- SpiderfyOnMaxZoom:
Wenn du auf einen Cluster an der unteren Zoomstufe klickst, spiderfy es so, dass du alle seine Marker sehen kannst. (Hinweis: Die Spiderfy tritt

bei der aktuellen Zoom-Ebene auf, wenn alle Elemente innerhalb des Clusters noch bei der maximalen Zoomstufe gruppiert sind oder mit dem Zoom, der durch die Option disableClusteringAtZoom angegeben wird).

- RemoveOutsideVisibleBounds: Cluster und Marker, die zu weit vom Viewport entfernt sind, werden für die Performance aus der Karte entfernt.
- Animieren: Glatte Split / Merge Cluster Kinder beim Zoomen und Spinnenfliegen. Wenn L.DomUtil.TRANSITION falsch ist, hat diese Option keine Wirkung (keine Animation ist möglich).

Zusätzlich können Sie noch die nachfolgenden Optionen verwenden.

- AnimateAddingMarkers:
Wenn auf true gesetzt (und animierte Option ist auch wahr), dann Hinzufügen einzelner Marker zur MarkerClusterGroup, nachdem sie der Karte hinzugefügt wurde, wird die Markierung hinzugefügt und sie in den Cluster animieren. Defaults auf false, da dies eine bessere Leistung bei der Markteinführung von Markierungen ergibt. AddLayers unterstützt dies nicht, nur addLayer mit einzelnen Markern.
- DisableClusteringAtZoom:
Wenn gesetzt, bei dieser Zoom-Ebene und unten, werden die Markierungen nicht gruppiert. Dies ist standardmäßig deaktiviert. Siehe Beispiel. Hinweis: Sie können daran interessiert sein, die Option spiderfyOnMaxZoom bei der Verwendung von disableClusteringAtZoom zu deaktivieren.
- MaxClusterRadius: Der maximale Radius, den ein Cluster von der zentralen Markierung (in Pixeln) abdecken wird. Default 80. Abnehmen wird mehr machen, kleinere Cluster. Sie können auch eine Funktion verwenden, die den aktuellen Kartenzoom annimmt und den maximalen Clusterradius in Pixel zurückgibt.
- PolygonOptions: Optionen, die beim Erstellen des L.Polygons (Punkte, Optionen) übergeben werden sollen, um die Grenzen eines Clusters anzuzeigen. Defaults leer, die Leaflet verwenden die Standard-Pfad-Optionen.
- SingleMarkerMode: Wenn auf true gesetzt, überschreibt das Symbol für alle hinzugefügten Marker, um sie als 1-Größen-Cluster erscheinen zu lassen. Hinweis: Die Marker werden nicht durch Clusterobjekte ersetzt, sondern nur ihr Icon wird ersetzt. Daher reagieren sie immer noch auf normale Ereignisse und die Option disableClusteringAtZoom stellt ihr vorheriges Icon nicht wieder her (siehe # 391).

- SpiderLegPolylineOptions: Ermöglicht es Ihnen, PolylineOptions zu spezifizieren, um Spinnenbeine zu stylen. Standardmäßig sind sie {weight: 1.5, color: '# 222', opacity: 0.5}.
- SpiderfyDistanceMultiplier: Erhöht von 1, um die Entfernung von der Mitte zu vergrößern, die spiderfied Marker platziert werden. Verwenden Sie, wenn Sie große Markersymbole verwenden (Standard: 1).
- IconCreateFunction: Funktion zum Erstellen des Cluster-Symbols. Siehe die Standardimplementierung oder das benutzerdefinierte Beispiel.
- ClusterPane: Kartenbereich, in dem die Cluster-Symbole hinzugefügt werden. Voreinstellung auf L.Markers Standard (aktuell 'markerPane'). Siehe das Fenster Beispiel.

Auch die Verarbeitung der Marker im Layer (todo wie Layer schreiben) können Sie beeinflussen. An Chunked AddLayers Optionen stehen Ihnen die nachfolgenden zur Verfügung.

- ChunkedLoading: Boolean, um die addLayers Verarbeitung in kleinen Intervallen zu teilen, so dass die Seite nicht einfriert.
- ChunkInterval: Zeitintervall (in ms), während dessen addLayers vor dem Pausieren arbeitet, um den Rest des Seitenprozesses zu lassen. Insbesondere verhindert dies, dass die Seite beim Einfügen vieler Markierungen einfriert. Standardmäßig auf 200ms.
- ChunkDelay: Zeitverzögerung (in ms) zwischen aufeinanderfolgenden Perioden der Verarbeitung für addLayers. Standard auf 50ms.
- ChunkProgress: Callback-Funktion, die am Ende jedes ChunkInterval aufgerufen wird. Typischerweise verwendet, um einen Fortschrittsindikator, z.B. Code in RealWorld 50k. Voreinstellung auf Null. Argumente:

Anzahl der verarbeiteten Marker

Gesamtzahl der hinzugefügten Marker

Verstrichene Zeit (in m)

Ereignissen

Leaflet-Ereignisse wie click() und mouseover() können Sie nur mit einem Marker im Cluster verwenden. Um Ereignisse auf ein Cluster anzuwenden, fügen Sie das Wort cluster vor den Namen des Ereignisses. Beispielsweise clusterclick oder clustermouseover. (todo noch testen)

```
markers.on('clusterclick', function (a) {
```

```
        console.log('cluster ' + a.layer.getAllChildMarkers().length);
    });
}
```

Zusätzlich können Sie noch folgende Ereignisse mit einem Cluster ausführen.

- Animation: Brände, wenn die Marker-Clustering / Unclustering-Animation abgeschlossen ist
- Spiderfied: Brände, wenn überlappende Marker spiderified werden (Enthält Cluster- und Markerattribute)
- Unspiderfied: Brände, wenn überlappende Markierungen nicht gespielt werden (Enthält Cluster- und Markerattribute) #

todo

Methoden

Hinzufügen und Entfernen von Markern können sie mit den Methoden `addLayer()` `removeLayer()` und `clearLayers()` werden unterstützt und sie sollten für die meisten Anwendungen arbeiten. (todo)

Das sichtbare Elternteil einer Markierung erhalten

Sie wenn Sie einen Marker in deiner MarkerClusterGroup hast und du das sichtbare Elternteil davon sehen möchtest (entweder selbst oder ein Cluster, in dem es sich befindet, ist derzeit auf der Karte sichtbar). Dies wird null zurückgeben, wenn der Marker und seine übergeordneten Cluster derzeit nicht sichtbar sind (sie sind nicht in der Nähe der sichtbaren Sicht)

```
var visibleOne = markerClusterGroup.getVisibleParent(myMarker);
console.log(visibleOne.getLatLng());
```

Marker animieren

Hüpfende Marker

<https://github.com/maximeh/leaflet.bouncemarker>

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
```

```

<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="bouncemarker.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
L.marker([50.27264, 7.26469],
{
bounceOnAdd: true,
bounceOnAddOptions: {duration: 5000, height: 100},
bounceOnAddCallback: function() {alert("Gelandet");}
})
.addTo(mymap);
</script>
</body>
</html>
<!--index_951.html-->

```

951.html

Als Erebnis sehen Sie einen Marker, der in die Karte springt. Nachdem der Marker gelandet ist, öffnet sich ein Pop-up-Fenster mit der Meldung: Gelandet!.

Animierte Marker

<https://github.com/openplans/Leaflet.AnimatedMarker>

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>

```

```

<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="AnimatedMarker.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 9);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var line = L.polyline(
[[50.68510, 7.94136],[50.68576, 6.94149],[51.68649, 6.94165]]),
animatedMarker = L.animatedMarker(line.getLatLngs(), {
distance: 2000,
interval: 1000,
});
mymap.addLayer(animatedMarker);
</script>
</body>
</html>
<!--index_950.html-->

```

950.html

Weitere Beispiel

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="AnimatedMarker.js"></script>
</head>
<body>
```

```

<button onclick="start()">Start</button>
<button onclick="stop()">Stop</button>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 9);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var line = L.polyline(
[[50.68510, 7.94136],[50.68510, 7.84136],[50.68510, 7.74136],
[50.68510, 7.64136],
[50.68510, 7.5136],[50.68510, 7.44136],[50.68510, 7.34136],
[50.68510, 7.24136]]),
animatedMarker = L.animatedMarker(line.getLatLngs(), {
autoStart: false,
distance: 200,
interval: 100
});
mymap.addLayer(animatedMarker);
function start(){animatedMarker.start();}
function stop(){animatedMarker.stop();}
</script>
</body>
</html>
<!--index_949.html-->
```

todo mit stop start 949.html

(<https://github.com/openplans/Leaflet.AnimatedMarker/issues/50>)

```

<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
```

```

<script src="AnimatedMarker.js"></script>
<script src="bouncemarker.js"></script>
</head>
<body>
<button onclick="start()">Start</button>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
b = new L.Marker([51.68649, 6.94165], {bounceOnAdd: true});
var line = L.polyline(
[[50.68510, 7.94136],[50.68576, 6.94149],[51.68649, 6.94165]]),
animatedMarker = L.animatedMarker(line.getLatLngs(), {
autoStart: false,
distance: 2000,
interval: 10,
onEnd: function() {
b.addTo(mymap);
b.bounce({duration: 100, height: 50});
mymap.removeLayer(animatedMarker);
setTimeout('mymap.removeLayer(b)', 900);
}
});
mymap.addLayer(animatedMarker);
function start(){animatedMarker.start();}
</script>
</body>
</html>
<!--index_948.html-->
```

todo 948.html bounce out Schlußfunktion

Leaflet Data Visualization Framework (DVF)

Das primäre Ziel des Frameworks ist es, die Datenvisualisierung und die thematische Kartierung mit Leaflet zu vereinfachen - so dass es einfacher ist, Rohdaten in überzeugende Karten zu verwandeln.

Todo <https://github.com/humangeo/leaflet-dvf> und
<https://github.com/humangeo/leaflet-dvf/wiki/1.0-Compatibility>

Das Plugin Data Visualization Framework

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js"></script>
<link rel="stylesheet" href="dvf.css" />
<script src="leaflet-dvf.js"></script>
<script src="leaflet-dvf.markers.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var marker = new L.MapMarker([50.27264, 7.26469],
{
radius: 30,
fillOpacity:0.5,
fillColor:'orange',
color:'purple',
innerRadius:7,
numberOfSides:4,
```

```

rotation:10
});
mymap.addLayer(marker);
var polygonmarker = new L.RegularPolygonMarker([50.27264,
6.26469],
{
numberOfSides: 3,
rotation: 10,
radius: 10,
fillColor:'green',
fillOpacity:1,
opacity:1,
weight:1,
radius:30
});
mymap.addLayer(polygonmarker);
var star = new L.StarMarker([50.27264, 8.26469],
{
numberOfPoints:8,
opacity:1,
weight:2,
fillOpacity:0,
radius:30});
mymap.addLayer(star);
</script>
</body>
</html>
<!--index_947.html-->
```

947.html

todo <https://github.com/humangeo/leaflet-dvf/wiki/6.-Markers>

Diagrammen als Marker

<https://github.com/humangeo/leaflet-dvf/wiki/6.-Markers> welche Marker es gibt mit bild.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.js"></script>
<link rel="stylesheet" href="dvc.css" />
<script src="leaflet-dvc.js"></script>
<script src="leaflet-dvc.markers.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 7);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var options = {
data: {
'data1': 20,
'data2': 50,
'data3': 10,
'data4': 20
},
chartOptions: {
'data1': {
fillColor: 'blue',
minValue: 0,
maxValue: 50,
maxHeight: 30,
displayText: function (value) {

```

```
//return value.toFixed(2);
return 'Mein Text';
},
},
'data2': {
fillColor: 'red',
minValue: 0,
maxValue: 50,
maxHeight: 30,
},
'data3': {
fillColor: 'green',
minValue: 0,
maxValue: 50,
maxHeight: 30,
},
'data4': {
fillColor: 'yellow',
minValue: 0,
maxValue: 50,
maxHeight: 30,
}
},
weight: 1,
color: '#000000',
radius:30,
fillOpacity:1
};

var bar = new L.BarChartMarker([50.27264, 7.26469], options);
mymap.addLayer(bar);

var radial = new L.RadialBarChartMarker([50.27264, 8.26469],
options);
mymap.addLayer(radial);

var pie= new L.PieChartMarker([50.27264, 6.26469], options);
mymap.addLayer(pie);

var cox = new L.CoxcombChartMarker([50.97264, 7.26469], options);
```

```

mymap.addLayer(cox);

var stack = new L.StackedRegularPolygonMarker([50.97264, 8.26469],
options);

mymap.addLayer(stack);

var radialmeter= new L.RadialMeterMarker([50.97264, 6.26469],
options);

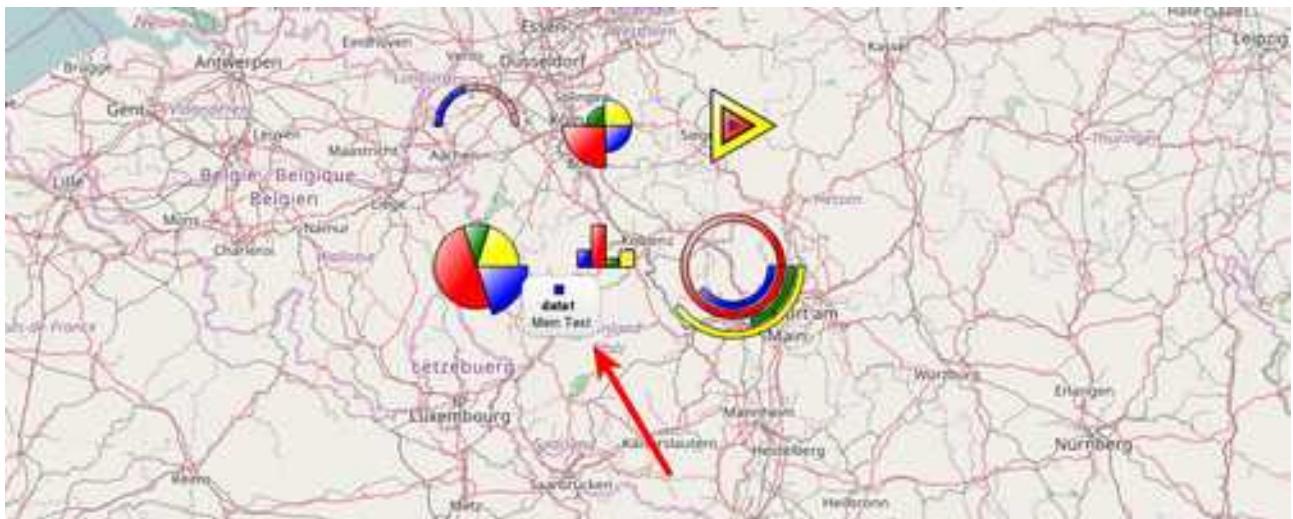
mymap.addLayer(radialmeter);

</script>
</body>
</html>

<!--index_946.html-->

```

946.html



972.png

todo Schluss

So, nun wissen Sie todo und können Daten visualisieren. Nun kann es sein, dass Sie selbst keine Datenbestände zum Anzeigen haben. Vielleicht bietet das ESRI Inc. (Environmental Systems Research Institute) Ihnen ja Daten, die zum Thema Ihrer Website passen? Im nächsten Kapitel erfahren Sie, wie Sie Daten dieses Institutes verwenden können.

In diesem Kapitel haben wir ...

todo

ESRI

Wenn Sie intensiver mit digitalen Landkarten arbeiten, möchten Sie sicherlich auch einmal Daten eines Geoinformationssystems auf Ihrer Karte anzeigen.

Geoinformationssysteme, Geographische Informationssysteme (GIS) oder Räumliche Informationssysteme (RIS) sind Informationssysteme zur Erfassung, Bearbeitung, Organisation, Analyse und Präsentation räumlicher Daten. Geoinformationssysteme umfassen die dazu benötigte Hardware, Software, Daten und Anwendungen. Todo umschreiben oder wikipedia link

Hierbei werden Sie früher oder später über den Begriff [Shapefile](#) stolpern. Dabei werden Sie auch das ESRI Inc. (Environmental Systems Research Institute) kennenlernen. Dieses Institut ist ein US-amerikanischer Softwarehersteller von Geoinformationssystemen (GIS) und hat unter anderem die Anwendung ArcGIS erstellt und verwendet als Datenformat unter anderem Shapefiles.

Todo <https://de.wikipedia.org/wiki/Shapefile>

gdb Geodatabase

<https://de.wikipedia.org/wiki/ArcGIS>

Aber auch wenn Sie nie auf den Begriff Shapefile stoßen werden Sie vielleicht einmal einen Webservice nutzen wollen, der der Endpunkt eines ARCServers ist.

Die wesentlichen Produkte sind Geoinformationssysteme und werden unter dem Namen ArcGIS als je eine Produktfamilie für den Server- und den Desktop-Bereich zusammengefasst. ArcGIS Desktop umfasst im Wesentlichen die Produkte ArcView, ArcEditor, ArcInfo (in den Komponenten ArcMap und ArcCatalog) sowie ArcReader und ArcGIS Explorer. Die Produktfamilie ArcGIS Server wird hingegen durch den ArcGIS Server in verschiedenen Editionen (Ausbaustufen), dem ArcIMS und der ArcSDE als zentrale Datenhaltungskomponente gebildet. Todo umschreiben oder link zu wikipedia

Bitte lassen Sie sich durch die vielen neuen Begriffe nicht verunsichern. Natürlich gibt es ein Leaflet Plugin, dass Ihnen beim Integrieren dieser Services zur Hand geht. Außerdem sehen wir uns alles Schritt für Schritt nacheinander an.

In diesem Kapitel werden wir ...

todo

<https://de.wikipedia.org/wiki/ESRI>

L.esri.BasemapLayer erweitert L.TileLayer

Mit der Klasse L.esri.BasemapLayer können Sie Esri gehostete Basemaps und Attribute Datenanbieter auf Ihrer Leaflet Karte anzuzeigen. Die [Nutzungsbedingungen](#) für Esri gehostete Dienste gelten für alle Leaflet-Anwendungen.

Todo Nutzungsbedingungen auch für anderes in kasten

Basemaps und optionale Layer von ESRI

Sie können Basemaps von ESRI verwenden toto link zu kapitel mit karten

Basemaps

Basemaps von ESRI bieten eine weltweite Abdeckung bei einer Vielzahl von Zoomstufen. Dabei können die folgenden Themenbereiche wählen.

- Streets
- Topographic
- NationalGeographic
- Oceans
- Gray
- DarkGray
- Imagery
- ShadedRelief
- Terrain
- USATopo (added in 2.0.0)

Die Basemap gefällt Ihnen grundsätzlich. Sie bietet Ihnen aber zu wenig Informationen. Dann legen Sie doch einfach einen der optionalen Layer mit den passenden Optionen über die Basemap.

Optionale Label Layer

Label Layer sind optionale Ebenen, die zusätzliche Textbeschriftungen zu den Basemaps hinzufügen. ESRI bietet Ihnen sieben verschiedene optionale Schichten:

- OceansLabels:
BESCHRIFTUNGEN, welche die Oceans Basemap ideal ergänzen.
- GrayLabels:
BESCHRIFTUNGEN, welche die Gray Basemap ideal ergänzen.
- DarkGrayLabels:
BESCHRIFTUNGEN, welche die DarkGray Basemap ideal ergänzen.
- ImageryLabels:
BESCHRIFTUNGEN, welche Ländergrenzen enthalten und die Imagery Basemap ideal ergänzen.
- ImageryTransportation:
BESCHRIFTUNGEN, welche Straßen enthalten und die Imagery Basemap ideal ergänzen.
- ShadedReliefLabels
BESCHRIFTUNGEN, welche die ShadedRelief Basemap ideal ergänzen.
- TerrainLabels
BESCHRIFTUNGEN, welche die Terrain Basemap ideal ergänzen.

Anzeigen von Basemaps bei höheren Auflösungen auf Retina-Geräten mit der Option `detectRetina`. Im nachfolgenden Programmcodeausschnitt wird die Basemap auf einem Retina Display (todo) in Retina-Auflösungen ausgegeben. Todo link zur Funktion in doku <https://esri.github.io/esri-leaflet/examples/retina-basemap.html> http://leafletjs.com/reference-1.2.0.html#tilelayer-detectretina

```
...
L.esri.basemapLayer('DarkGray', {
  detectRetina: true
}) .addTo(map);
...
```

Ein einfaches Beispiel

<http://www.esri.com/data/basemaps>

todo <https://github.com/Esri/esri-leaflet> (CDN?)

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet-debug.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 7);
var gray = L.esri.basemapLayer('Gray').addTo(mymap);
var graylabels = L.esri.basemapLayer('GrayLabels').addTo(mymap);
gray.setOpacity(0.5);
gray.on('load', alertme);
function alertme(){
alert("Fertig!");
}
</script>
</body>
</html>
<!--index_945.html-->
```

945.html

Das haben wir gemacht

(todo alert regagiert auch wenn ich karte verschiebe)

Das ist Ergebnis mit uns ohne label



943.png



944.png

todo was muss ich tun

todo hier alle basemap layer und label layer: <https://esri.github.io/esri-leaflet/api-reference/layers/basemap-layer.html>

Switch Basemaps

<https://esri.github.io/esri-leaflet/examples/switching-basemaps.html>

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet-debug.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<select
style ="position: absolute; top: 10px; right:10px; z-index: 400; "
id="basemaps"
onchange="changeBasemap(basemaps)"
>
<option value="Topographic">Topographic</option>
<option value="Streets">Streets</option>
<option value="NationalGeographic">National Geographic</option>
<option value="Oceans">Oceans</option>
<option value="Gray">Gray</option>
<option value="DarkGray">Dark Gray</option>
<option value="Imagery">Imagery</option>
<option value="ShadedRelief">Shaded Relief</option>
<option value="Terrain">Terrain</option>
<option value="USATopo">USATopo </option>
</select>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 7);
var layer = L.esri.basemapLayer('Gray').addTo(mymap);
function changeBasemap(basemaps) {
var basemap = basemaps.value;
```

```
if (layer) {  
    mymap.removeLayer(layer);  
}  
  
layer = L.esri.basemapLayer(basemap);  
mymap.addLayer(layer);  
}  
</script>  
</body>  
</html>  
<!--index_944.html-->
```

944.html

Was haben wir gemacht?

Das ist das Ergebnis:



942.png Wenn Imager ausgewählt ist sieht das so aus.

Shapefiles

Das Dateiformat Shapefile (oft Shapedaten oder Shape genannt) ist ein ursprünglich für die Software ArcView der Firma ESRI entwickeltes Format für Geodaten.

Was genau verbirgt sich hinter dem Begriff Shapefiles

Ein Shapefile ist keine einzelne Datei. In der Regel handelt es sich um mehrere Dateien, die zu einem Zip-Archiv zusammengefasst sind. In diesem Archiv müssen mindestens die drei nachfolgend genannten Dateitypen vorhanden sein:

- Eine Datei mit der Endung `.shp` dient zur Speicherung der Geometriedaten.
- Eine Datei mit der Endung `.dbf` enthält Attributdaten im dBASE-Format.
- Eine Datei mit der Endung `.shx` dient als Index. Mithilfe dieser Datei sind die Geometrie mit den Attributdaten in der `.dbf`-Datei verknüpft.

todo

Wie kommen Sie an ein Shapefiles

Todo natürlich können Sie selbst erstellen. Das ist aber nicht Thema dieses Buches.

todo Shapefiles findet man, wenn man nach open data und einer Stadt sucht.

Todo open data koblenz shapefile hat

https://lvermgeo.rlp.de/fileadmin/lvermgeo/pdf/open-data/Download_von_Verwaltungsgrenzen.pdf ergeben und hier zum Beispiel Flurgrenzen: http://geo5.service24.rlp.de/wfs/verwaltungsgrenzen_rp.fcgi?VERSION=1.1.0&SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=fluren_rlp&SRNAME=EPSG:25832&outputFormat=SHAPEZIP

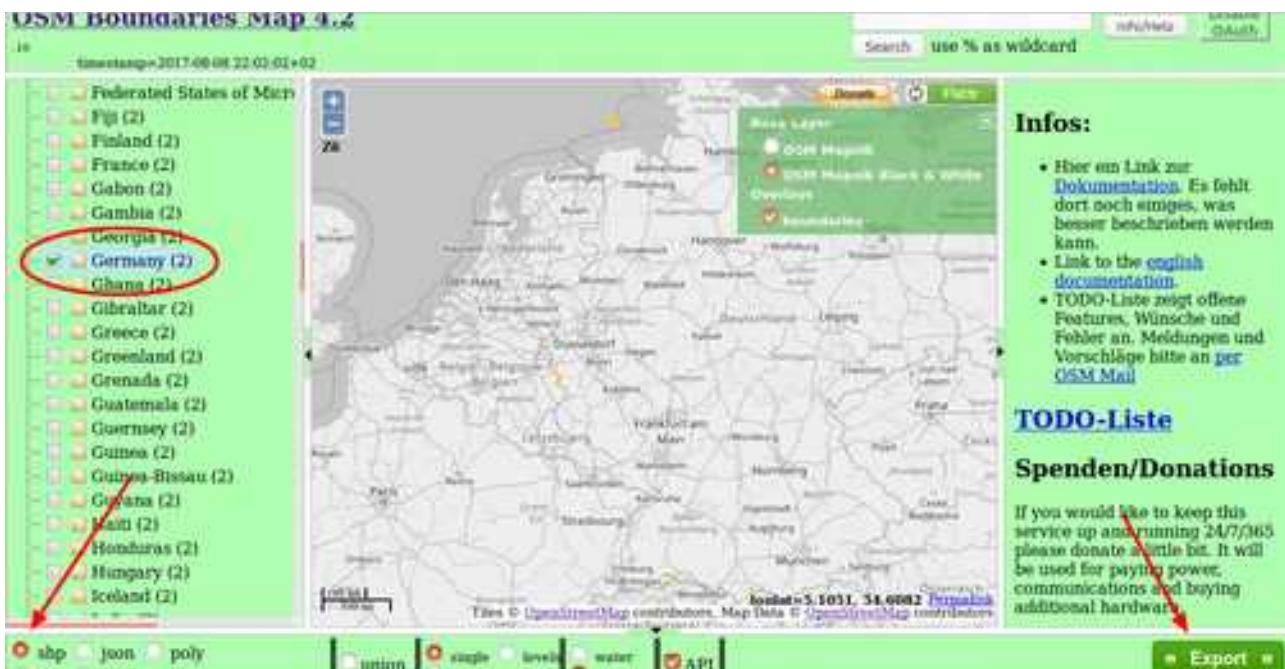
The screenshot shows a file manager window with the title 'result (1).zip'. The interface includes standard buttons for 'Entpacken' (Extract) and '+'. Below the title bar is a toolbar with icons for back, forward, home, and search. The main area displays a list of files with columns for Name, Größe (Size), Typ (Type), and Geändert (Changed). The list contains four items:

| Name | Größe | Typ | Geändert |
|----------------|-----------|-----------------|--------------------------|
| fluren_rlp.shx | 268,0 kB | ESRI-Shape... | 30. November 1979, 00:00 |
| fluren_rlp.shp | 89,2 MB | ESRI-Shape... | 30. November 1979, 00:00 |
| fluren_rlp.prj | 388 Bytes | ESRI-Koordin... | 30. November 1979, 00:00 |
| fluren_rlp.dbf | 22,7 MB | Xbase-Dok... | 30. November 1979, 00:00 |

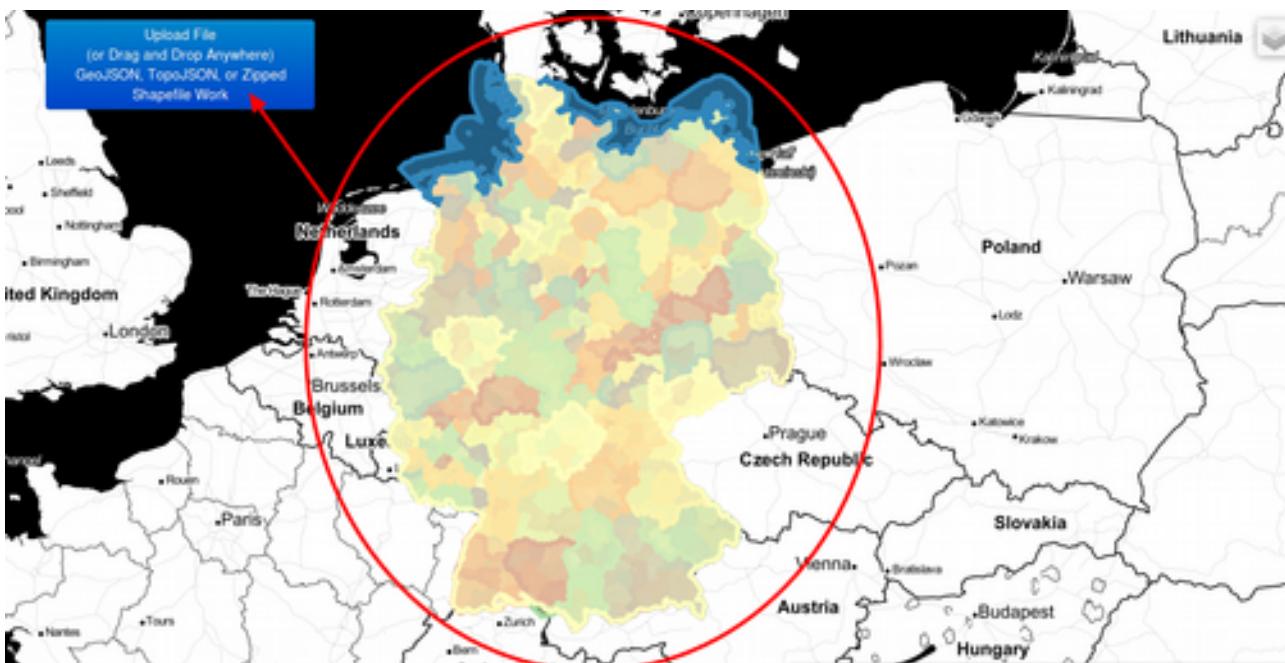
Oder <http://wiki.openstreetmap.org/wiki/Shapefiles>

bietet weltweit Shapefiles aller administrativen Grenzen. Die Datenbank wird um Mitternacht aktualisiert. Erstellt von einem aktiven Mitglied im deutschen OSM-Forum.

todo <https://wambachers-osm.website/boundaries/idx42o.jsp> todo bild



971.png



todo testen 969.png <http://leaflet.calvinmetcalf.com/#6/51.358/11.964>

Wie binden Sie Shapefiles in Ihre Leaflet Karte ein?

Deutsche Verwaltungsgrenzen als Shapefile in der Leaflet Karte

Mein Beispiel <https://www.arcgis.com/home/item.html?id=ae25571c60d94ce5b7fcf74e27c00e0>

The screenshot shows the ArcGIS item page for a dataset titled "Verwaltungsgrenzen Deutschland (De, Länder, Rgbz, Kreise)". The page includes a map preview showing administrative boundaries of Germany, a description of the dataset, and download options. A red arrow points to the "Shapefile" link in the description section.

Datensatz mit 4 Shape Dateien mit den Verwaltungsgrenzen von Deutschland
von MOONSHINE
Zuletzt geändert: 28. Dezember 2011
Shapefile

Download

Beschreibung

Details

Erstellt: 27. Dezember 2011
Größe: 1 MB

Besitzer

970.png

Datei mit Namen `exported_boundaries_Germany.shp.zip` ist im Downloadverzeichnis.

Das Archiv enthält die Dateien

-

todo leaflet shape file plugin :

<https://github.com/calvinmetcalf/leaflet.shapefile> 2 dateien

`L.Shapefile = L.GeoJSON.extend({`

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8" />
<title>Eine OSM Karte mit Leaflet</title>
```

```
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 6);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var shpfile = new L.Shapefile('vg2500_geo84.zip');
shpfile.addTo(mymap);
</script>
</body>
</html>
<!--index_943.html-->
```

943.html

Das haben wir gemacht

Das ist das Ergebnis.



967.png

Deutsche Verwaltungsgrenzen mit Optionen

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 6);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var shpfile = new L.Shapefile('vg2500_geo84.zip',
{style:function(feature){return
{color:"black",fillColor:"red",fillOpacity:.75}}})
);
shpfile.addTo(mymap);
</script>
</body>
</html>
<!--index_942.html-->
```

942.html



968.png

todo vielleicht shape files ansehen

todo optionen des plugins leaflet shape ansehen.

Ein Pop-up-Fenster

Todo wie popup

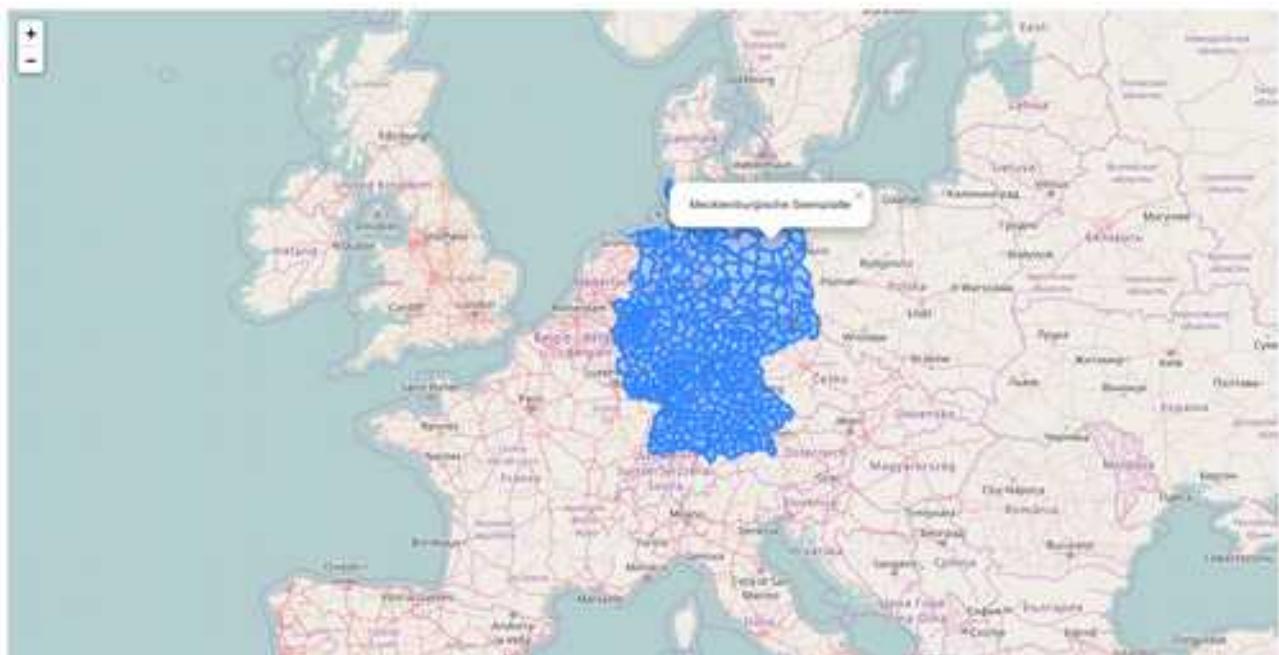
```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
```

```
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 5);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var shpfile = new L.Shapefile(
'vg2500_geo84.zip',
{
onEachFeature:function(feature, layer)
{
layer.bindPopup(feature.properties.GEN);
}
});
shpfile.addTo(mymap);
</script>
</body>
</html>
```

Was haben wir gemacht

Das ist das Ergebnis

941.html



966.png

Ein Pop-up-Fenster mit allen Informationen des Features

Tip 940.html falls du features nicht kennst.

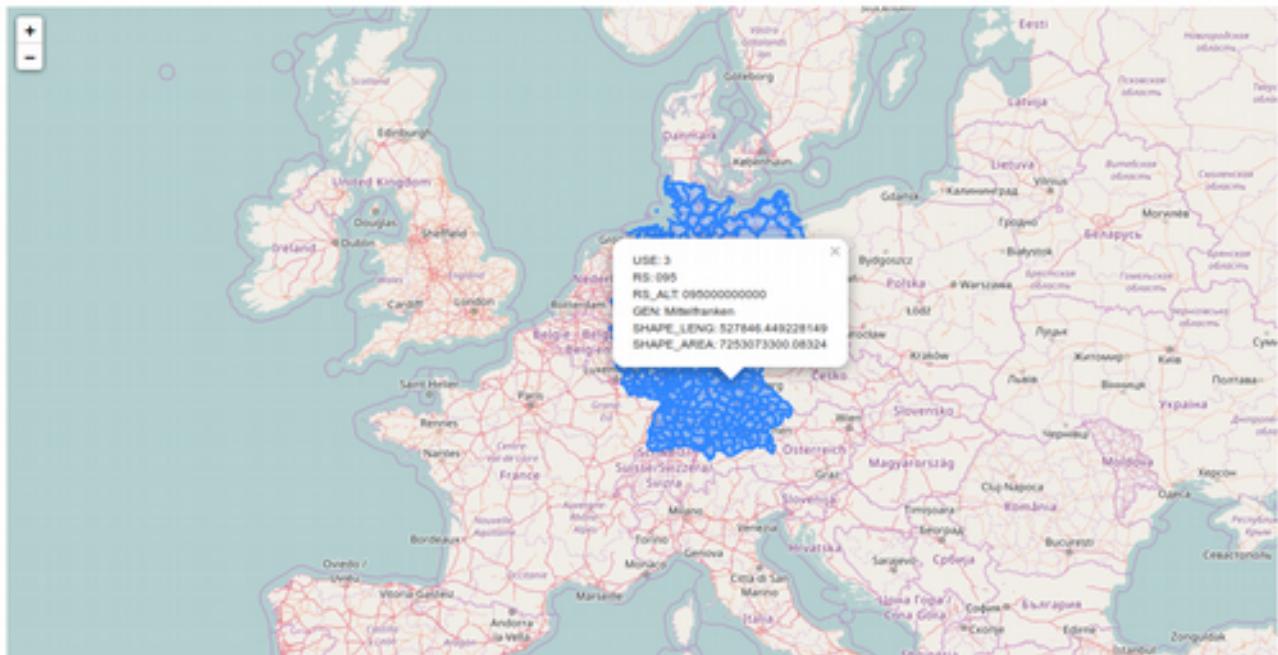
```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="shp.js"></script>
<script src="leaflet.shpfile.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 5);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
var shpfile = new L.Shapefile(
'vg2500_geo84.zip',
{
onEachFeature:function(feature, layer)
{
var holder=[];
for (var key in feature.properties){
holder.push(key+": "+feature.properties[key]+"<br>");
popupContent=holder.join("");
layer.bindPopup(popupContent);
}
} );
shpfile.addTo(mymap);
</script>
```

```
</body>
</html>
<!--index_940.html-->
```

940.html

Was haben wir gemacht

Das ist das Ergebnis



965.png

ESRI Services

Sie wissen nun wie Sie eine ESRI Basiskarte laden, wie Sie mit dem Plugin ESRI-leaflet umgehen und wie Sie das Dateien im Format shapefile nutzen können.

ESRI bietet aber noch mehr. Sie können Webservices nutzen. Das ESRI Plugin unterstützt Sie auch bei der Verbindung mit einem Geografischen Webservice.

Ganz nebenbei Base noch können Sie sieben weitere Basemaps nutzen.

- L.esri.BasemapLayer
- L.esri.DynamicMapLayer
- L.esri.ImageMapLayer
- L.esri.RasterLayer

- L.esri.TiledMapLayer
- L.esri.FeatureLayer
- L.esri.Cluster.FeatureLayer
- L.esri.Heat.FeatureLayer

<http://esri.github.io/esri-leaflet/api-reference/>

L.esri.DynamicMapLayer

Todo? <https://github.com/Esri/esri-leaflet>

<http://esri.github.io/esri-leaflet/api-reference/layers/dynamic-map-layer.html>

<https://geoportal.stadt-koeln.de/arcgis/rest/services/Behindertenparkplaetze/MapServer/0>

<https://geoportal.stadt-koeln.de/arcgis/rest/services/>

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.97264, 7.00469], 12);
//L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/
{y}.png').addTo(mymap);
L.esri.basemapLayer('Gray').addTo(mymap);
var url = "https://geoportal.stadt-
koeln.de/arcgis/rest/services/Behindertenparkplaetze/MapServer";
var dMap = L.esri.dynamicMapLayer({
url: url,
```

```
opacity : 0.25,  
useCors: false  
}).addTo(mymap);  
dMap.bindPopup(  
function(err, featureCollection, response){  
return featureCollection.features[0].properties.Bezeichnung +  
'<br>Anzahl: ' +  
featureCollection.features[0].properties.Anzahl;  
});  
</script>  
</body>  
</html>  
<!--index_939.html-->
```

939.html

Das haben wir gemacht

Das ist das Ergebnis:



938.png

todo ich kann auch andere Karte tile layer nehmen

geocoding

Was ist Geocoding? Geocoding bezeichnet die Umwandlung von Adressen wie „Kirchstraße 13, 56571 Muster“ in geografische Koordinaten wie „50.423021

Breite und -7.083739 Länge“, die Sie zum Platzieren von Markern auf einer Karte oder zum Positionieren der Karte verwenden können (<https://developers.google.com/maps/documentation/geocoding/intro?hl=de>).

<https://de.wikipedia.org/wiki/Georeferenzierung>

todo es bit auch

<http://wiki.openstreetmap.org/wiki/Nominatim> → nicht so sicher, man kann aber daten eintragen!

Und <https://developers.google.com/maps/documentation/geocoding/start>

<https://github.com/Esri/esri-leaflet-geocoder>

Nach Eingabe einer Adresse den passenden Ort auf der Karte finden
Umgekehrtes Geocoding bezeichnet die Umwandlung geografischer Koordinaten in lesbare Adressen.

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<link rel="stylesheet" href="esri-leaflet-geocoder.css">
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.97264, 7.00469], 12);
```

```
L.esri.basemapLayer('Gray').addTo(mymap);

var searchControl = L.esri.Geocoding.geosearch().addTo(mymap);

var results = L.layerGroup().addTo(mymap);

searchControl.on("results", function(data) {
  results.clearLayers();
  for (var i = data.results.length - 1; i >= 0; i--) {
    results.addLayer(L.marker(data.results[i].latlng));
  }
});

</script>
</body>
</html>
<!--index_938.html-->
```

938.html

Achtung: Adresse nur im Bereich wird gefunden

Das haben wir gemacht

Das ist Ergebnisse



937.png

Mithilfe eines Parameters in der URL den passenden Ort finden

```
<!DOCTYPE HTML>
```

```

<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var x = location.search;
var y = x.split("=");
var temp=y[1];
var address = decodeURIComponent(temp);
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.97264, 7.00469], 3);
L.esri.basemapLayer('Gray').addTo(mymap);
L.esri.Geocoding.geocode().text(address).run(function(err, result,
response) {
console.log(result.results[0].latlng);
L.marker(result.results[0].latlng).addTo(mymap);
mymap.setView(result[0].latlng,13);
});
</script>
</body>
</html>
<!--index_937.html-->

```

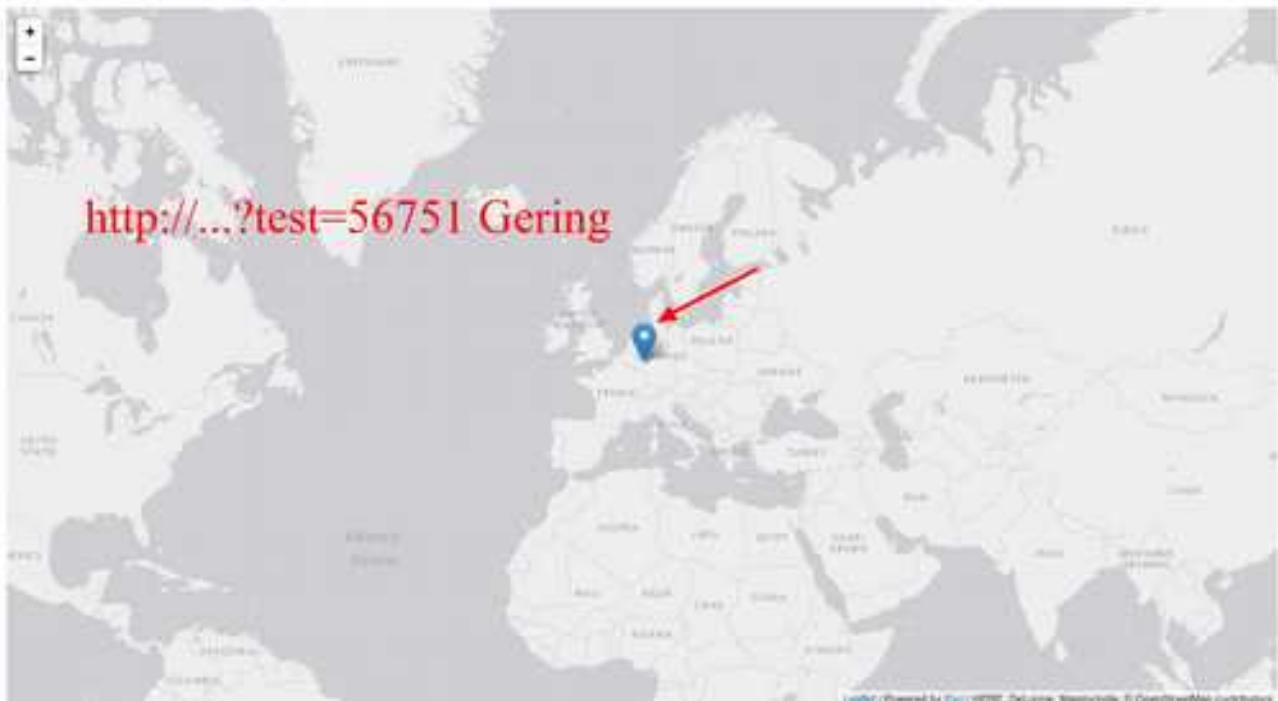
937.html

<http://esri.github.io/esri-leaflet/api-reference/tasks/geocode.html>

Das haben wir gemacht:

<file:///media/astrid/AstridsAktuelleDatenPlatte/Daten/arbeit/Schreiben/bookbon/softwaretesting/BuchLeaflet/code/5/URLgeocode.html?a=56751%20Gering>

Das ist Ergebnis



936.png

Nach Klick auf die Karte die passende Adresse anzeigen

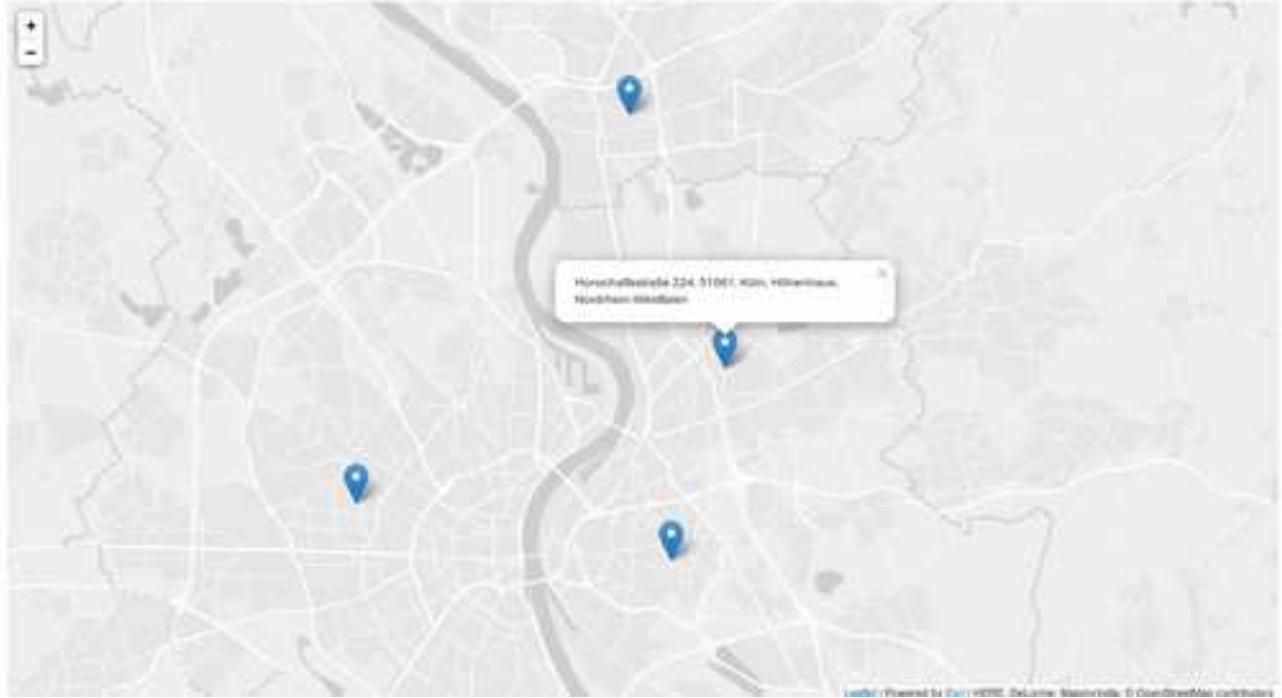
```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
<script src="esri-leaflet-geocoder.js"></script>
</head>
<body>
<div style="height: 700px;" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.97264, 7.00469], 12);
L.esri.basemapLayer('Gray').addTo(mymap);
```

```
mymap.on('click', function(e) {
var r = L.marker();
L.esri.Geocoding.reverseGeocode()
.latlng(e.latlng)
.run(function(error, result, response) {
console.log(result.address);
mymap.removeLayer(r);
r =
L.marker(result.latlng).addTo(mymap).bindPopup(result.address.Match_
h_addr).openPopup();
});
});
});
</script>
</body>
</html>
<!--index_936.html-->
```

936.html

Das haben wir gemacht

Das ist Ergebnisse



935.png

<http://esri.github.io/esri-leaflet/api-reference/tasks/reverse-geocode.html>

query layers

Todo Punkte im ausgewählten Bereich werden gefunden. Deshalb Bereich klein halten, falls es viele Punkte gibt.

Wir haben im letzten Kapitel Daten über einen Webservice geladen. Dabei haben immer alle Daten auf der Karte angezeigt. Das kann sehr unübersichtlich werden. Meist bietet ein Webservice ja große Datenmengen an. Sie möchten sicherlich nur die für Sie und Ihre Anwender relevanten Daten auf Ihrer Karte anzeigen. Eine Lösung für die Aufgabe erarbeiten wir in diesem Kapitel.

Todo unterschied feature server map server:

<https://gis.stackexchange.com/questions/126373/difference-between-feature-server-and-map-server>

doku feature layer: <http://esri.github.io/esri-leaflet/api-reference/layers/feature-layer.html>

todo noch gucken : <https://gis-iq.esri.de/openstreetmap-daten-aus-deutschland-in-arcgis-online/>

todo was gibt es in GIS: <https://gis-iq.esri.de/content-auf-der-arcgis-plattform-viel-mehr-als-nur-basemaps/>

Attribut

Konfigurieren einer Abfrage zum Filtern von Funktionen auf einem

[L.esri.FeatureLayer](#).

<http://esri.github.io/esri-leaflet/examples/feature-layer-popups.html>

sicherheitseinrichtungen: <http://www.arcgis.com/home/item.html?id=59711f4ee839438299c90164115f129b>

```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
```

```

<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
</head>
<body>
<select id="sicherheitseinrichtungenID">
    <option value="">Reset</option>
    <option
value="Untertyp='pt_Notrufsaeule'">Notrufsaeule</option>
    <option value="Untertyp='pt_Feuerwache'">Feuerwache</option>
    <option
value="Untertyp='pt_Rettungspunkt'">Rettungspunkt</option>
    <option
value="Untertyp='pt_Polizeistation'">Polizeistation</option>
    <option value="Untertyp='pt_Sirene'">Sirene</option>
    <option
value="Untertyp='pt_Rettungswache'">Rettungswache</option>
</select>
<div style="height: 700px" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 12);
L.esri.basemapLayer('Gray').addTo(mymap);
var url =
'http://services.arcgis.com/OLiydejKCZTGhvWg/arcgis/rest/services/
OSM_DE_Sicherheitseinrichtungen/FeatureServer/0';
var sicherheitseinrichtungen =
document.getElementById('sicherheitseinrichtungenID');
var sicherheitseinrichtungenLayer = L.esri.featureLayer({
url: url
}).addTo(mymap);
var popupTemplate = "<h3>Details:</h3>\n\
Land: {Land}<br>\n\
Kreis: {Kreis}<br>\n\
Gemeinde: {Gemeinde}<br>\n\

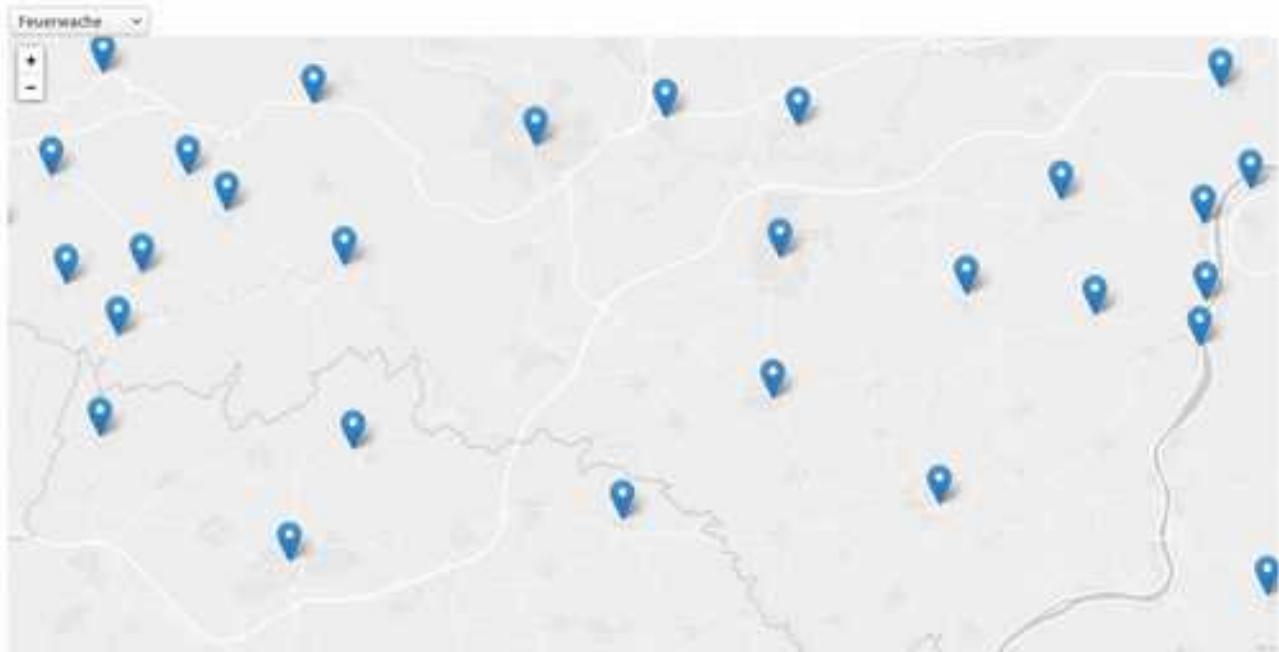
```

```
Stadt: {Stadt} <br>\n\
Typ: {Typ}, {Untertyp}";
sicherheitseinrichtungenLayer.bindPopup(function (layer) {
console.log(layer.feature.properties);
return L.Util.template.popupTemplate, layer.feature.properties)
});
sicherheitseinrichtungen.addEventListener('change', function(){
sicherheitseinrichtungenLayer.setWhere(sicherheitseinrichtungen.value);
});
</script>
</body>
</html>
<!--index_935.html-->
```

935.html

Das haben wir gemacht

Das ist ergebnis



933.png Alle Feuerwehrwachen im Bereich

todo: Osm Daten sind nicht immer vollständig.

abstand

Ausführen von erweiterten Abfragen auf einer Feature-Ebene. Klicken Sie auf die Karte, um die Funktionen innerhalb von 500 Metern abzufragen. Weitere Informationen über Feature Layer finden Sie in der Dokumentation:
<http://esri.github.io/esri-leaflet/api-reference/layers/feature-layer.html>.
(<http://esri.github.io/esri-leaflet/examples/querying-feature-layers-2.html>)

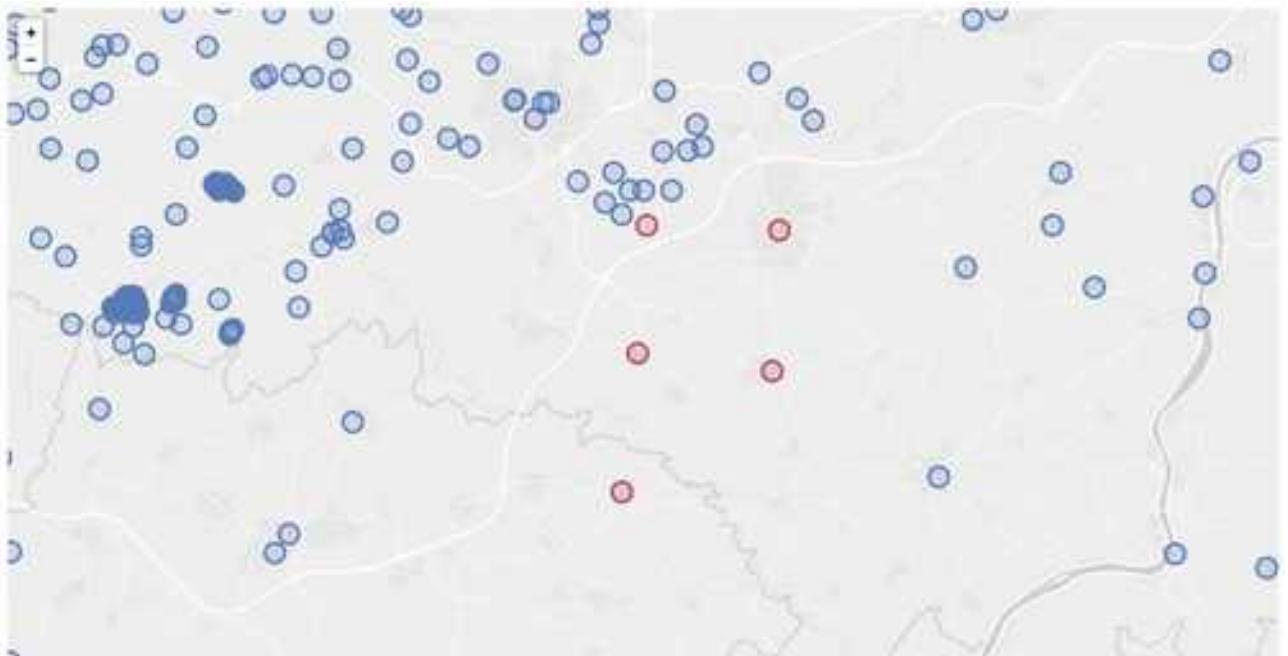
```
<!DOCTYPE HTML>
<html lang="de">
<head>
<meta charset="utf-8"/>
<title>Eine OSM Karte mit Leaflet</title>
<link rel="stylesheet" href="../leaflet/leaflet.css" />
<script src="../leaflet/leaflet.js"></script>
<script src="esri-leaflet.js"></script>
</head>
<body>
<div style="height: 700px" id="mapid"></div>
<script>
var mymap = L.map('mapid',
{doubleClickZoom:false}).setView([50.27264, 7.26469], 12);
L.esri.basemapLayer('Gray').addTo(mymap);
var url =
'http://services.arcgis.com/OLiydejKCZTGhvWg/arcgis/rest/services/
OSM_DE_Sicherheitseinrichtungen/FeatureServer/0 ';
var sicherheitseinrichtungen =
document.getElementById('sicherheitseinrichtungenID');
var sicherheitseinrichtungenLayer = L.esri.featureLayer({
url: url,
pointToLayer: function (geojson, latlng) {
return L.circleMarker(latlng);
},
style:{
color: '#5B7CBA',
}
}).addTo(mymap);
```

```
var popupTemplate = "<h3>Details:</h3>\n\
Land: {Land}<br>\n\
Kreis: {Kreis}<br>\n\
Gemeinde: {Gemeinde}<br>\n\
Stadt: {Stadt} <br>\n\
Typ: {Typ}, {Untertyp}";\n\
sicherheitseinrichtungenLayer.bindPopup(function (layer) {\n\
return L.Util.template(popupTemplate, layer.feature.properties)\n});\n\
var nearbyIds = [];\n\
mymap.on('click', function(e) {\n\
sicherheitseinrichtungenLayer.query().nearby(e.latlng,\n\
5000).ids(function(error, ids) {\n\
for (var j = 0; j < nearbyIds.length; j++) {\n\
sicherheitseinrichtungenLayer.resetStyle(nearbyIds[j]);\n\
}\n\
nearbyIds = ids;\n\
for (var i = 0; i < ids.length; i++) {\n\
sicherheitseinrichtungenLayer.setFeatureStyle(ids[i], {\n\
color: '#BA454E',\n\
});\n\
}\n\
});\n\
});\n</script>\n</body>\n</html>\n<!--index_934.html-->
```

934.html

Das haben wir gemacht

Das ist Ergebnis



934.png Klick färbt alle Punkte die im Bereich sind

In diesem Kapitel haben wir ...

todo

Leaflet benutzerfreundlich in einem Joomla! Modul

In diesem Kapitel werden wir ...

Todo

Leaflet versucht nicht, jedem alles recht zu machen. Stattdessen konzentriert sich das JavaScript Framework darauf, das was Sie zum Erstellung einer Karte benötigen, perfekt anzubieten. Ganz unabhängig davon, wie Sie die Karte präsentieren. Das hat Vorteile. Das Framework bietet eine flexible Schnittstelle und Sie haben viele Möglichkeiten. Nachteilig ist, dass Sie sich selbst um alles darum herum kümmern müssen. Heute werden Karten meist online angezeigt. Ein beliebtes Content Management System, mit dem viele Websites gepflegt werden, ist [Joomla!](#) Warum nehmen Sie nicht einfach Joomla! als Drumherum.

Beispielhaft zeige ich Ihnen hier, wie Sie Ihre mit Leaflet erstellte Karte mithilfe eines Moduls in Joomla! benutzerfreundlich einsetzen können und wie Sie es anderen bedienerfreundlich ermöglichen, benutzerdefinierte Marker in die Karte einzufügen.