Projektbericht

LV: 716409 VU Geoinformatik: Web mapping

Lektoren: B.Sc.. Klaus Förster, Mag. Bernhard Öggl

SS 2022

Freizeit in der Stadt Wien
Winter- und Sommerangebote

Bearbeiter	Maria Heinrich	Mnr: 12143466
	Paula Spannring	Mnr:
Abgabedatum	29. Juni 2022	
	Universität Innsbruck	

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Ursprüngliche Ideen und Probleme

Die ursprüngliche Idee war, auf Basis des Tiroler Lawinenreports (https://lawinen.report/bulletin/latest) eine Karte zu erstellen, die für die einzelnen Warnregionen die Warnstufe abhängig vom Lawinenproblem oder von der Exposition oder von der Höhe anzeigt. GeoJSON – Daten für die Regionen sind vorhanden, ebenso ist der jüngste Report als XML – Datei verfügbar. Es war möglich, das die XML-Datei in Javascript zu importieren, jedoch war es (für uns in kurzer Zeit zur Konzeptabgabe) nicht möglich, die nötigen Daten aus dem importierten XML zu extrahieren.

Daraufhin war die nächste Idee, für das Salzburger – Land Sommer- und Winteraktivitäten in jeweils einer Karte darzustellen. Es gibt Datensätze in https://www.data.gv.at/ , jedoch waren nicht als open-source-Dateien verfügbar und somit für das Vorhaben nutzlos.

2 Konzepterstellung

2.1 Idee

Da im Kurs bereits mit Inhalten aus Wien gearbeitet wurde, wurde die Idee von Salzburg auf die Stadt Wien übertragen. Eine Startseite sollte einen Überblick geben und auf eine Aktivitätenseite verweisen. Da diese Idee vergleichsweise einfach scheint, wurden zwei untergliederte Seiten erstellt, eine Sommer- und eine Winterseite.

2.2 Vorgehen

Da zwei Bearbeiterinnen am Projekt beteiligt sind, soll jede eine Aktivitätenseite erstellen. Darüber hinaus ist die Startseite zu erstellen und verlinken.

3 Erstellung der Webseiten

3.1 Startseite

Die Startseite gibt einen kleinen Einblick und wird auf Basis der im Kurs erstellten NZ-Etappen aus den ersten Sitzungen aufgebaut. Dieses Gerüst, wofür die „main.css“ adaptiert und angepasst wird, stellt auch das Gerüst für die Winter- und Sommerseiten dar. Dafür wird ein „main.css“-Datei angelegt, worauf auch aus den anderen Seiten zugegriffen wird. Der Unterschied zur Startseite ist, dass der Aufruf mit „../main.css“ geschieht, statt lediglich mit „/main.css“, da auf einen anderen Ordner zugegriffen wird. Allerdings müssen die Leaflet-Plugins mit dem „Stylesheet“ in jeder HTML-Seite als Plugins aufgerufen werden.

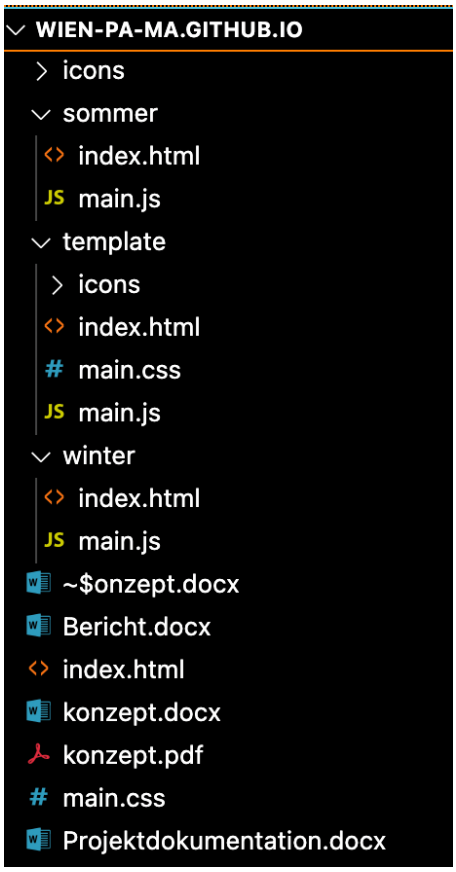

Abbildung 1: Struktur mit main.css im "Hauptordner"

```
<!-- Font awesome (Style-Sheet) einbauen - Link aus dem
Internet über cdnjs.com gedownloaded
integrity: stellt sicher, das der Inhalt der Datei genauso
ist, wie die Checksumme besagt - kein Virus kann eingebaut
werden
crossorigin/refarrerpolicy - vom Anbieter, nciht
interessant für uns-->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/
ajax/libs/font-awesome/6.1.1/css/all.min.css"
    integrity="sha512-KfkfwYDsLkIlwQp6LFnl8zNdLGxu9YAA1QvwIN
    ks4PhcElQSvqcyVLLD9aMhXd13uQjoXtEKNosOWaZqXgel0g=="
    crossorigin="anonymous" referrerpolicy="no-referrer" />
```

Abbildung 2: PlugIn für Stylesheet

3.2 Basisseite für Sommer und Winter mit Karte

3.2.1 Index.html

Die html – Datei enthält im head – Bereich den Link zum Wiener Wappen als Shotcut Icon, den Titel, die Verlinkung zum jeweiligen Javascript und CSS - Stylesheet und die Implementierung der Plugins, diese werden später genauer beschrieben.

Im body – Bereich wird im Header mit Wiener Wappen, Überschrift und Link zum Github-Repository (der jeweiligen Seite) angezeigt, im article – Bereich werden die Karte aufgerufen und die Links zur Startseite und zur jeweils anderen Aktivitäten-Seite angegeben. Die Bilder werden über die Open-Souce-Site https://pixabay.com/de/ bezogen.

Im footer werden die verwendeten Daten (als Liste) verlinkt.

3.2.2 Main.js

Die Karte wird nach Implementieren des Leaflet – Plugins in dem div – Container des HTML eingefügt. Das Zentrum der Karte soll Wien sein, die Zoomstufe, und der Default – Layer (Openstreetmap) wird festgelegt.

Folgende Leaflet – Plugins sind (nach Implementieren in der HTML – Datei) der Karte zugefügt:

- ein Maßstab,
- einen Button zum Wechseln auf Fullscreen,
- eine Minimap,

- einen Button, um nach Zoomen und die Karte verschieben, wieder die Ausgangsposition der Karte (Koordinaten und Zoomstufe) zu erreichen,
- einen Button, um die Entfernung verschiedener Punkte messen zu können.

Außerdem soll in einem Layer- und Overlay – Control zwischen mehreren Layer und Overlays (Aktivitäten je nach Saison) ausgewählt werden können.

3.3 Inhalt für Winter

Als Overlays sollen verschiedene Möglichkeiten für Aktivitäten in Wien im Winter dargestellt werden. Die Kategorien, die im Layercontrol an- und ausgeschaltet werden können, sind:

- Schwimmbäder
- Silvesterpfad
- Sportstätten

Zum Darstellen der Layer werden in asynchronen Funktionen die GeoJSON – Daten mittels URL geladen:

```
let response = await fetch(url);
let geojson = await response.json();
```

Die jeweiligen Funktionen werden dann mit dem jeweils passenden URL zu den GeoJSON -Daten (siehe Quellenangaben) aufgerufen.

In diesen Daten sind nur Punkte. Diese werden mit Marker und passenden Popups and die overlays gefügt und mit passendem Icon dargestellt. Die Icons sind in einem extra Ordner gespeichert.

```
L.geoJSON(geojson, {
      pointToLayer: function (geoJsonPoint, latlng) {
      return L.marker(latlng, {
                  icon: L.icon({
                       iconUrl: "../icons/swimming2.png",
                      iconAnchor: [16, 37],
                      popupAnchor: [0, -37]
                  })
              }).bindPopup(popup);
          }
    }).addTo(overlays.schwimmen);
```

3.3.1 Schwimmbäder

Die Schwimmbäder haben ein Attribut „AUSLASTUNG_AMPEL_KATEGORIE_0“, wenn dieses den Wert 1 hat, ist das Schwimmbad geöffnet, und es sind noch Plätze verfügbar. Um nur geöffnete Schwimmbäder mit freien Plätzen anzuzeigen, wurden diese durch eine if – Abfrage selektiert:

```
let offen = geoJsonPoint.properties.AUSLASTUNG_AMPEL_KATEGORIE_0;
          if (offen == 1) {…}
```

innerhalb der if – Abfrage wird der Marker zurückgegeben.

Im Popup werden Name, Adresse und Webseite des Schwimmbads angegeben:

```
let popup = `
            <strong> ${geoJsonPoint.properties.NAME} </strong>
            <hr>
            Adresse: ${geoJsonPoint.properties.ADRESSE}<br>
            <a href="${geoJsonPoint.properties.WEBLINK1}">Weblink</a>
            `;
```

3.3.2 Silvesterpfad

In Wien gibt es einen Silvesterpfad mit verschiedenen Stationen des Pfads, Toiletten und Erste-Hilfe Stationen. Der Datensatz enthält Punkte für jeden Stand.

Die Stände werden als Marker mit Popup angezeigt, der Bezeichnung, Beschreibung und der Link zur Webseite des Standes (falls vorhanden) beinhaltet. Je nach Typ der Station wird der Marker als jeweiliges Icon dargestellt (Toilette, Erste Hilfe, Stand des Silvesterpfads).

Hierfür wird eine Variable für den Typ erstellt (Typ = 1,2,3 für Pfad-Station, Toilette, Erste Hilfe), die Icons wurden jeweils silvester_1(2,3) und so die passenden Icons für die Stände erstellt:

```
let typ = geoJsonPoint.properties.TYP
            return L.marker(latlng, {
                icon: L.icon({
                    iconUrl: `../icons/silvester_${typ}.png`,..
                        })
                  })
```

Die Stationen des Silvesterpfads sind durchnummeriert (1-12), diese sollten in der richtigen Reihenfolge durch eine Polylinie verbunden werden. Hierfür mussten zuerst die features in die richtige Reihenfolge gebracht werden: das Attribut „Bezeichnung“ enthält (wenn der Typ gleich „Stand“ ist, es sich also um eine Station des Silvesterpfads handelt) die Nummer der Station am Anfang der Bezeichnung. Also konnte man die features nach der Bezeichnung für die richtige Reihenfolge sortieren. Da die Nummern bis 12 gingen, und die Bezeichnung ein String ist, wurde im nächsten Schritt der String gesplittet, sodass nur nach der Nummer (als Typ Integer nach Umwandlung) sortiert werden konnte. Somit war die Reihenfolge für die durchnummerierten Stände richtig. Ohne das zweite Sortieren wäre die Reihenfolge: 1 – 10 – 11 – 12 – 2 – 3 - …

```
// features sortieren, dass alle Stationen mit Nummer vorne sind
geojson.features.sort(function (a, b) {
  return a.properties.BEZEICHNUNG.split("-")[0] >
      b.properties.BEZEICHNUNG.split("-")[0];
    });
// features nach Nummer sortieren
geojson.features.sort(function (a, b) {
```

```
        return parseInt(a.properties.BEZEICHNUNG.split("-")[0]) >
            parseInt(b.properties.BEZEICHNUNG.split("-")[0]);
    });
```

Wie zuvor angemerkt, ist die Geometrie der Daten Punkte, es sollen jedoch eine Polylinie dargestellt werden, wofür ein Array mit den Koordinaten benötigt wird. Die Funktion „ArrayFromPoints“ speichert für die features, für die der Typ Stand ist (selektiert mit if-Abfrage), die Koordinaten als Array in eine neue Variable „stations“:

```
function ArrayFromPoints(geojson) {
        let stations = []
        for (i = 0; i < geojson.totalFeatures; i += 1) {
            if (geojson.features[i].properties.TYP == 1) {
                stations.push([geojson.features[i].geometry.coordinates[1],
                        geojson.features[i].geometry.coordinates[0]])

            }
        }
        return stations
    }
```

Die neue Variable „stations“ kann dann als Polylinie dargestellt werden, die die durchnummerierten Stände in der richtigen Reihenfolge miteinander verbindet. Sie wird ebenso wie die Marker zu den Ständen an das Overlay „Silvesterpfad“ gebunden:

```
let polyline = L.polyline(ArrayFromPoints(geojson), {
        color: '#ad59c2'
    }).addTo(overlays.silvester);
```

3.3.3 Sportstätten

Im Datensatz sind sowohl indoor - als auch outdoor - Sportstätten enthalten. Es sollten jedoch nur indoor - Sportstätten dargestellt werden. Hierzu werden diese durch eine if – Abfrage (entspricht Attribut „KATEGORIE_NUM“ = 2) selektiert. Außerdem sollten passende Icons zu den Sportarten, die in den jeweiligen Sportstätten betrieben werden, den Marker in der Karte darstellen. Die Sportarten sind im Attribut „SPORTSTAETTEN_ART“ beschrieben. So kann durch eine Erweiterung der if – Abfrage selektiert werden, ob die Sportart, die mit dem passenden Icon dargestellt werden soll enthalten ist:

```
let kategorie = geoJsonPoint.properties.KATEGORIE_NUM;
let art = geoJsonPoint.properties.SPORTSTAETTEN_ART;
if (kategorie == 2 & art.includes('Tennis') || kategorie == 2 &
            art.includes('Badminton')) {
                return L.marker(latlng, {
                    icon: L.icon({
```

```
                    iconUrl: "../icons/tennis.png",
                     . . .
```

In diesem Fall werden Badminton- und Tennis – Sportstätten mit dem gleichen Icon dargestellt.

Eigentlich sollte eine Funktion erstellt werden, die zwei Argumente als String haben sollte: die Sportart, nach der selektiert wird und die Bezeichnung des icons. Diese sollte dann für alle unterschiedlichen Sportarten aufgerufen werden, etwa so:

```
function IconKategorie(bezeichung, icon){
                if (kategorie == 2 & art.includes(bezeichnung)) {
                    //console.log(geoJsonPoint.properties)
                    return L.marker(latlng, {
                        icon: L.icon({
                            iconUrl: "../icons/icon.png",
                            iconAnchor: [16, 37],
                            popupAnchor: [0, -37]
                        })
                    }).bindPopup(popup);
                };
            }
IconKategorie("Tennis", "tennis")
```

So wurde jedoch kein Marker mit Icon in der Karte dargestellt. Eine Vermutung ist, dass die Funktion IconKategorie ein weiteres „return“ Element benötigt. Nach vielen fehlgeschlagenen Versuchen wird wie zuvor beschrieben die if – Abfrage mit Marker – Erstellung mit passendem Icon kopiert und für verschiedene Sportarten (Kletterhalle, Eishalle, Tennis, Fußball, Handball, Bowling) verwendet.

3.4 Inhalt für Sommer

3.4.1 Allgemeines

Im Sommer eignen sich die Daten zur Lokalisierung von Spielplätzen, Waldspielplätzen, Parkanlagen, Badestellen, Grillzonen an der Donau und Fußgängerzonen in Wien zum Aufenthalt und Entspannen. Die Daten werden als .json-Dateien zur Verfügung bereitgestellt und unterscheiden sich primär dadurch, dass es sich entweder um Feature-Points, oder -Polygons handelt. Feature-Points dem Befehl ‚geoJson-Point.‘ die Feature-Polygone als ‚“ frei gewählte variable (hier „layer“)“.feature.‘ und zugehörigen Pfad (in diesem Falle stets ‚properties‘) und dem benötigten Attribut aufgerufen.

Aufruf	**Pfad .properties**	**Attribut**	**Beispiel (Ausgabe)**
`// Badestellen` `Name/Standort: ${geoJsonPoint.properties.BEZEICHNUNG}`			Name/Standort: **Neue Donau, Höhe Segelhafen, rechts**

//Grillzonen Lage: ${layer.feature.properties.LAGE}	Lage: 21., Neue Donau, linkes Ufer, zwischen Brigittenauer Brücke und Brigittenauer Badebucht,

Tabelle 1: Aufruf von Points & Polygonen aus .json-Dateien

```
Badestellen                                                    main.js:106
▼Object i
  ▶crs: {type: 'name', properties: {…}}
  ▼features: Array(28)
    ▼0:
      ▶geometry: {type: 'Point', coordinates: Array(2)}
        geometry_name: "SHAPE"
        id: "BADESTELLENOGD.1582781"
      ▼properties:
          ANZ_ECOLI: 15
          ANZ_ENTEROKOKKEN: 46
          BADEQUALITAET: 1
          BEZEICHNUNG: "Neue Donau, Höhe Segelhafen, rechts"
          BEZIRK: 21
          SE_ANNO_CAD_DATA: null
          SE_SDO_ROWID: 1582781
          SICHTTIEFE: 2.1
          TYP: 1
          UNTERSUCHUNGSDATUM: "2022-06-06Z"
          WASSERTEMPERATUR: 23.3
          WEITERE_INFO: "http://www.wien.gv.at/forschung/laboratorien/umweltmedizin/wasse
        ▶[[Prototype]]: Object
        type: "Feature"
      ▶[[Prototype]]: Object
    ▶1: {type: 'Feature', id: 'BADESTELLENOGD.1582782', geometry: {…}, geometry_name: '
```

Abbildung 3: Beispiel der .json-Datei der Badestellen in Wien.
Array mit 28 Features (Badestellen). geometry Point - Punktförmiger Aufruf mit ‚GeoJsonPoint.properties. …". Icon als Anzeigeicon (Quelle: Bildschirmfoto des Aufrufs (console.log(„Badestellen", geojson) nach Aufruf der .json-Datei mit async function loadBaden(url) {let response = await fetch(url);let geojson = await response.json(); …loadBaden("https://data.wien.gv.at/daten/geo?service=WFS&request=GetFeature&version=1.1.0&typeName=ogdwien:BADESTELLENOGD&srsName=EPSG:4326&outputFormat=json"))

```
Grillzonen:                                                    main.js:227
▼Object i
  ▶crs: {type: 'name', properties: {…}}
  ▼features: Array(3)
    ▼0:
      ▶geometry: {type: 'Polygon', coordinates: Array(1)}
        geometry_name: "SHAPE"
        id: "GRILLZONEOGD.fid--207a3c5b_1818aea0f40_6c03"
      ▼properties:
          LAGE: "21., Neue Donau, linkes Ufer, zwischen Brigittenauer Brücke und Brigittena
          OBJECTID: 7361
          RESERVIERUNG: "nein"
          SE_ANNO_CAD_DATA: null
          WEBLINK1: "https://www.wien.gv.at/umwelt/gewaesser/donauinsel/freizeit/grillen.ht
        ▶[[Prototype]]: Object
        type: "Feature"
      ▶[[Prototype]]: Object
```

Abbildung 4: Aufbau des .json für Grillzonen. 3 Arrays entsprechen 3 Flächen mit Grillplätzen. type = Polygon, aufgerufen mit Layer

(Quelle: Bildschirmfoto des Aufrufs (console.log(„Grillzonen", geojson) nach Aufruf der .json-Datei mit async function loadGrill(url) {let response = await fetch(url);let geojson = await response.json(); …

loadGrill("https://data.wien.gv.at/daten/geo?service=WFS&request=GetFeature&version=1.1.0&typeName=ogd-wien:GRILLZONEOGD&srsName=EPSG:4326&outputFormat=json"))

3.4.2 Point-Features

Am Beispiel der Badestellen wurde der Aufbau der Inhalte der Features, die als Punkte abgespeichert sind, erläutert (s. oben). Dazu zählen die Spielplätze, Waldspielplätze, Parkanlagen und Badestellen. Diese unterscheiden sich primär durch ihre Darstellung mittels Marker (Icon) und Popup-Inhalte (s. Tabelle 2).

	Waldspielplätze	**Spielplätze**	**Parkanlagen**	**Badestellen**
Popup-Inhalt (Attribute)	• Standort • Bezirk • Angebot • Weblink	• Name • Bezirk • Spielplatzelement	• Name • Flächengröße • Hunde erlaubt • Spielen erlaubt • Wasser vorhanden • Öffnungszeiten • Weblink	• Name • Wassertemperatur • Wasserqualität
../icon/"-„	spielplatz.png	wspielplatz.png	urbanpark.png	spielplatz.png

Tabelle 2: Popup-Inhalte und Icons der GeoJsonPoint-Features im Sommer

Zunächst werden die Daten geladen und als geojson ausgegeben. Dieses wird mit ‚L.goJSON(geojson)' aufgerufen und der Variablen geojson übergeben. Anschließend wird ein durch Koordinaten, die aus der .json-Datei herausgelesen werden, definierter Punkt in der Karte als geoJsonPoint definiert und mit Informationen bestückt. Dazu wird ein Popup definiert und beschrieben. Der aktuelle Punkt wird mit ‚geoJsonPoint' definiert. Anschließend wird mit dem Schlüsselwort ‚properties' der Ursprungsdatei der Pfad aufgerufen und die Informationen der jeweiligen Attribute herausgelesen.

```
async function loadBaden(url) {
    let response = await fetch(url);
    let geojson = await response.json();
    console.log("Badestellen ", geojson); //nur ums in der Console zu sehen
    L.geoJSON(geojson, {
        pointToLayer: function (geoJsonPoint, latlng) {
            let popup = `
                Name/Standort:        <br><strong>${geoJsonPoint.properties.BEZEICH-
NUNG}</strong>
                <hr>
                Wassertemperatur: ${geoJsonPoint.properties.WASSERTEMPERATUR}<br>
                Wasserqualität: ${geoJsonPoint.properties.BADEQUALITAET}<br>
                <a   href="${geoJsonPoint.properties.WEITERE_INFO}"   target="_blank"
>Weblink</a>
        `;
```

Entsprechend der Inhalte werden Icons auf https://mapicons.mapsmarker.com/ herausgesucht, farblich im Ordner „Icons" abgespeichert und über das Keyword ‚iconUrl:' mit „../icons/"-".png" aufgerufen. Diese Icons stellen je nach Kategorie/Aktivität den jeweiligen Marker dar. Die Popups werden mit einem „return L.marker", wie im unten beschriebenen Aufruf für Parkanlagen, mittels einer Variablen an den Punkt, der durch die mit Variablen beschriebene Koordinaten ‚latlng' verortet wird, zurückgegeben und dem Layer der Karte hinzugefügt:

```
return L.marker(latlng, {
                icon: L.icon({
                    iconUrl: `../icons/baden.png`,
```

```
                    iconAnchor: [16, 37],
                    popupAnchor: [0, -37]
                })
            }).bindPopup(popup);
        }
    }).addTo(overlay.baden);
}
```

Das Icon kann mittels weiterer Keywords formatiert werden, z. B. können Icon oder Popup verschoben werden. Anschließend werden Marker samt Popup, das im Marker gespeichert ist, an die Karte übergeben.

Zusammengefasst wird zunächst das Popup definiert, an einen Marker übergeben und dieser Marker wird samt Popup an den Layer übergeben.

Zum Schluss wird die url aufgerufen, dies geschieht jedoch bereits am Anfang des Codes:

```
loadBaden("https://data.wien.gv.at/daten/geo?service=WFS&request=GetFeature&ver-
sion=1.1.0&typeName=ogdwien:BADESTELLENOGD&srsName=EPSG:4326&outputFormat=json");
```

Für Inhalte mit vielen Punkten bzw. Stationen auf der Karte wurden für eine übersichtlichere Darstellung MarkerCluster als PlugIns installiert. Dafür wird im index.html das Leaflet-Plugin implementiert und in den jeweiligen Stellen (Spielplätze & Parkanlagen) im main.js aufgerufen.

Geschlossen werden

3.4.3 Polygon-Features

Flächenfeatures ähneln Punkfeatures. Ein großer unterschied ist, dass zunächst die Flächen als Layer in die Karte eingebaut werden und die Flächen innerhalb der Funktion mit einem Popup versehen werden. Dafür werden zunächst alle Daten geladen, anschließend erfolgt der geoJSON-Aufruf durch eine Funktion mit der Variablen feature. Die Flächen werden durch ‚return' an die Variable übergeben. Anschließend wird dem Feature ein Popup zugewiesen. Anders als bei Punkten geschieht dies nicht über einen Marker, sondern über die Fläche. Die Fläche mitsamt Popup werden durch .addTo(overlay.'layername') an den Layer übergeben, der anfangs der Karte hinzugefügt wurde.

Die beiden Inhalte Fußgängerzonen und „Grillplätze" unterscheiden sich lediglich in der Variablen „loadZones" und „loadGrill" und dem Aufruf der URL, der Farbe (color) und dem Inhalt des Popups. Die Grillplätze beinhalten im Popup deren Lage, die Möglichkeit zu reservieren und einen Link zu weiterführenden Informationen.

```
async function loadZones(url) {
    let response = await fetch(url);
    let geojson = await response.json();
    L.geoJSON(geojson, {
        style: function (feature) {
            return {
                color: "#F012BE",
                weight: 1,
```

```
                opacity: 0.9,
                fillOpacity: 0.5,
            }
        }
    }).bindPopup(function (layer) {
        return `
        <h4>Fußgängerzone ${layer.feature.properties.ADRESSE}</h4>
        <p>Zeitraum: ${layer.feature.properties.ZEITRAUM || ""}</p>
        <p>${layer.feature.properties.AUSN_TEXT || ""}</p>
        `;
    }).addTo(overlay.fussgaenger);
}
loadZones("https://data.wien.gv.at/daten/geo?service=WFS&request=GetFeature&ver-
sion=1.1.0&typeName=ogdwien:FUSSGEHERZONEOGD&srsName=EPSG:4326&outputFormat=json");
```

3.4.4 Plugin „MarkerClusterGroup“

Das Plugin wird für Spielplätze und Parkanlagen verwendet. Im Folgenden wird am Beispiel der Parkanlagen der Aufbau und Aufruf erläutert.

Zunächst wird das Leaflet-Plugin im index.html eingebunden für main.css und main.js. Anschließend im main.js aufgerufen. Dafür wird eine zusätzliche Variable eingeführt.

```
let cparkanlagen = L.markerClusterGroup({
        disableClusteringAtZoom: 17
    });
    cparkanlagen.addTo(overlay.parkanlagen);
```

Wichtig ist hierbei, dass alle Inhalte später auf die Variable „cparkanlagen“ gespeichert werden (Marker- und Popup-Aufruf), da der Aufruf ansonsten nicht die Inhalte clustert.

```
L.geoJSON(geojson, {
        pointToLayer: function (geoJsonPoint, latlng) {
            let popup = `…`;
            return L.marker(…).bindPopup(popup);
        }
    }).addTo(cparkanlagen); //Markercluster
    }).addTo(overlay.parkanlagen); //kein Markercluster
```

4 Quellenangaben

Da es sich um einen Vorgehensbericht handelt, wird von einer für Facharbeiten üblichen Literaturangabe abgesehen. Alle verwendeten Quellen sind auf den oben beschriebenen Websites verlinkt. Folgend eine Auflistung der verwendeten Quellen zur Erstellung und Inhaltsfüllung der Websites. Die Seiten wurden zuletzt am 22.06.2022 auf ihre Aktualität geprüft.

Plugins

- Leaflet: https://leafletjs.com/examples/quick-start/
- Leaflet – providers: https://cdnjs.com/libraries/leaflet-providers
- Fullscreen: https://cdnjs.com/libraries/leaflet.fullscreen
- Minimap: https://cdnjs.com/libraries/leaflet-minimap
- Reset view: https://github.com/drustack/Leaflet.ResetView
- Polyline measure: https://github.com/ppete2/Leaflet.PolylineMeasure
- MarkerClusterGroup: https://github.com/Leaflet/Leaflet.markercluster

Font awesome Stylesheet

- Stylesheets: https://cdnjs.com/

Daten

- Schwimmbäder: https://www.data.gv.at/katalog/dataset/stadt-wien_schwimmbderstandorte-wien/resource/29041605-d50d-4696-8827-bb496ac9ad75
- Silvesterpfad: https://www.data.gv.at/katalog/dataset/f5d7a560-0599-4ee6-b9c1-5efec380b912
- Sportstätten: https://www.data.gv.at/katalog/dataset/71084c02-973d-4544-b804-7ed82bd53027
- Badestellen: https://www.data.gv.at/katalog/dataset/stadt-wien_badestellenstandortewien/resource/53195ad6-3c6d-47f9-b56f-b42e4e8af83c
- Waldspielplätze: https://www.data.gv.at/katalog/dataset/waldspielplatze-wien/resource/e942f7d6-16c9-4fda-9bbe-bd547f516160
- Spielplätze: https://www.data.gv.at/katalog/dataset/spielplatze-standorte-wien/resource/6f27a91a-bb1e-44f4-a683-98cb80f63379
- Grillzonen: https://www.data.gv.at/katalog/dataset/grillzonen-der-stadt-wien/resource/2341ba7e-15cf-41f9-abf4-e76d6c317eb0
- Parkanlagen: https://www.data.gv.at/katalog/dataset/parkanlagen-standorte-wien/resource/e2aecfe2-9862-4178-a43c-8c994d676782
- Fußgängerzonen: https://www.data.gv.at/katalog/dataset/stadt-wien_fugngerzonenwien/resource/8972d745-60cc-47c5-9d42-a28a995de177

Icons & Images

- Icons: https://mapicons.mapsmarker.com/
- Colors: https://clrs.cc/
- Images: https://pixabay.com/de/

Sonstige

- GitHub webmapping: https://github.com/webmapping