

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: 11702687

Abgabedatum

29. Juni 2022

Universität Innsbruck

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	II
1 Ursprüngliche Ideen und Probleme.....	Fehler! Textmarke nicht definiert.
2 Konzepterstellung.....	1
2.1 Idee	<i>Fehler! Textmarke nicht definiert.</i>
2.2 Vorgehen und Aufgabenverteilung	1
3 Erstellung der Webseiten.....	1
3.1 Startseite	1
3.2 Basisseite für Sommer und Winter mit Karte	2
3.2.1 Index.html.....	2
3.2.2 Main.js.....	3
3.3 Inhalt für Winter	3
3.3.1 Schwimmbäder.....	4
3.3.2 Silvesterpfad.....	4
3.3.3 Sportstätten.....	5
3.4 Inhalt für Sommer	6
3.4.1 Allgemeines.....	6
3.4.2 Point-Features.....	7
3.4.3 Polygon-Features.....	9
3.4.4 Plugin „MarkerClusterGroup“	10
4 Quellenangaben	12

Abbildungsverzeichnis

Abbildung 1: Struktur mit main.css im "Hauptordner"	2
Abbildung 2: PlugIn für Stylesheet (cdnjs.cloudflare.com → google-fonds)	2
Abbildung 3: unterschied der font-awesome-stylesheets von cdnjs.cloudflare und google-fonds (die Unterschiede entstehen aufgrund verschieden gewählter Schriftarten und müssten nicht entstehen. Für das Projekt werden Google-Fonds verwendet).....	2
Abbildung 4: Beispiel der .json-Datei der Badestellen (type Point) in Wien.	7
Abbildung 5: Aufbau des .json für Grillzonen (type Polygon). 3 Arrays entsprechen 3 Flächen mit Grillplätzen. type = Polygon, aufgerufen mit Layer.....	7
Abbildung 6: Karte der Sommerseite in Web-Ansicht. Quelle: eigene Darstellung.....	8
Abbildung 7: links: Marker ohne Gruppierung; rechts: Gruppierung und Anzeige mehrerer Marker; .11	

Tabellenverzeichnis

Tabelle 1: Aufruf von Points & Polygonen aus .json-Dateien	6
Tabelle 2: Popup-Inhalte und Icons der GeoJsonPoint-Features im Sommer.....	8

1 Themenfindung und -entwicklung

Ursprünglich sollte auf Basis des Tiroler Lawinenreports (<https://lawinen.report/bulletin/latest>) eine Karte erstellt, die für tiroler Regionen die Lawinenwarnstufe anzeigt. Dabei sollte eine Filterung nach Lawinenproblem, Exposition und Höhe möglich sein. Für die Region ist der jüngste Report als XML-Dateien verfügbar. Da laut Aufgabenstellung allerdings GeoJSON-Formate importiert werden sollten, diese jedoch nicht verfügbar waren, wurde die Idee verworfen.

Stattdessen sollten für das Land Salzburg Freizeitaktivitäten in einer interaktiven Karte dargestellt werden. Wie in der Vorlesung wurden Datensätze im GeoJSON-Format der Seite <https://www.data.gv.at/> bezogen. Allerdings konnten deren Inhalte nicht geöffnet werden, da es sich um verschlüsselte Dateien handelte. Sie waren folglich für das Vorhaben nutzlos.

Im Kurs wurde bereits mit Inhalten aus Wien gearbeitet. Folglich liegt es nahe, die Idee von Salzburg auf die Stadt Wien zu übertragen. Eine Startseite soll einen Überblick geben und auf eine Aktivitätsseite verweisen. Da dieses Konzept vergleichsweise einfach scheint, wird das Projekt in zwei saisonale Seiten untergliedert. Paula Spannring erstellt die Winter-, Maria Heinrich erstellt die Sommerseite.

1.1 Vorgehen und Aufgabenverteilung

Im Weiteren sieht die Aufgabenverteilung wie folgt aus.

Paula Spannring:

- identisches Gerüst für Kartenapplikation (main.js) – Winter- und Sommerseite
- Einbau der identischen PlugIns der Winter- und Sommerseite
- Bestückung der Winterseite
- Erstellung des Berichts

Maria Heinrich

- Startseite mit Bildern, Links und Stylesheets
- Stylesheet, Schriften & main.css im Hauptordner und Aufruf auf den Unterseiten
- Strukturierung: Unternehmen im GitHub, Ordnerstruktur (Icons, Images, main.css für alle Seiten)
- Dokumentation: Erstellung des Konzepts, Erstellung der Präsentation
- Layout & Formalitäten des Berichts
- Bestückung der Sommerseite

2 Erstellung der Webseiten

2.1 Startseite (Maria Heinrich)

Die Startseite gibt einen kleinen Einblick und wird auf Basis der im Kurs erstellten NZ-Etappen aus den ersten Sitzungen aufgebaut. Das Gerüst und die Art und Weise der Styles, wofür die „main.css“ adaptiert und angepasst wird, bietet auch die Basis für die Winter- und Sommerseiten (s. Abbildung 1). Dafür wird ein „main.css“-Datei angelegt, worauf auch aus den anderen Seiten zugegriffen wird. Der Unterschied zur Startseite, deren index.html-Datei im selben Hauptordner liegt, ist, dass der Aufruf mit „./main.css“ geschieht, statt lediglich mit „/main.css“, da auf einen anderen Ordner zugegriffen wird.

Damit Icons unter den Bildern dargestellt werden, müssen die Font-awesome-Styles der Seite <https://cdnjs.com/libraries/font-awesome> eingebaut werden. Für die Kartenapplikationen müssen

Leaflet-Stylesheets implementiert werden. Um noch andere Schriften zu beziehen, werden im main.css Google-Fonds eingebaut, die im auch hier aufgerufen werden (s. Abbildung 2). Im main.css können auf dieser Basis u. a. verschiedene Schriftarten für Abschnitte oder Textarten festgelegt, Seitenränder definiert und die Seite vom Hintergrund abgesetzt werden.

The screenshot shows a GitHub repository structure for 'WIEN-PA-MA.GITHUB.IO'. The 'main.css' file is highlighted. The code in main.css includes meta tags for charset, viewport, and edge compatibility, followed by a title, links to 'main.css' and 'font-awesome' via CDNJS, and a font import rule for Google Fonts.

```

EXPLORER ...  

GEÖFFNETE EDITOREN  

WIEN-PA-MA.GITHUB.IO   

  > anderes  

  > icons  

  > images  

  sommer  

    index.html  

    main.js  

  winter  

    index.html  

    main.js  

  index.html  

# main.css

```

```

index.html ./ X index.html sommer # main.css
index.html > html > body > main > header > div > img
<head>
  <meta charset="UTF-8">
  <link rel="shortcut icon" href="../images/wappen.png" type="image/png">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Freizeit in Wien</title>
  <link rel="stylesheet" href="main.css">
  <!-- Font awesome (Style-Sheet) einbauen - für alle Seiten-->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
  font-awesome/6.1.1/css/all.min.css"
        integrity="sha512-KfkfwYdsLkIlwQp6LFnl8zNdLGxu9YAA1QvwINKs4PhcElQSvqcyVLLD
        9aMhxd13u0jOXTENos0WaZqXgel0g==" crossorigin="anonymous" referrerPolicy="no-referrer" />
</head>
@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@300&
family=Open+Sans:ital@0;1&family=Roboto:wght@300&display=swap');

```

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

Abbildung 2: PlugIn für Stylesheet (cdnjs.cloudflare.com → google-fonds)

The comparison highlights differences in the font-awesome stylesheets used in the header of the pages. The top version uses a CDNJS link, while the bottom version uses a Google Fonts import rule.

Sommer- und Winteraktivitäten in Wien

Starttext

Willkommen in Wien. Auf folgenden Seiten können verschiedene Freizeitangebote für ein aktives Leben in und um Wien erkundet werden. Dafür kann je nach Jahreszeit in Winter oder Sommer unterschieden werden. Viel Spaß beim erkunden!

Links zu Sommer- und Winteraktivitäten

- [Winteraktivitäten in und um Wien](#)
- [Sommeraktivitäten in und um Wien](#)

Sommer- und Winteraktivitäten in Wien

Starttext

Willkommen in Wien. Auf folgenden Seiten können verschiedene Freizeitangebote für ein aktives Leben in und um Wien erkundet werden. Dafür kann je nach Jahreszeit in Winter oder Sommer unterschieden werden. Viel Spaß beim erkunden!

Links zu Sommer- und Winteraktivitäten

- [Winteraktivitäten in und um Wien](#)
- [Sommeraktivitäten in und um Wien](#)

Abbildung 3: Unterschied der font-awesome-stylesheets von cdnjs.cloudflare und google-fonds (die Unterschiede entstehen aufgrund verschiedener Schriftarten und müssten nicht entstehen. Für das Projekt werden für die Schriften Google-Fonds verwendet, für Bildunterschriften Fond-Awesome-Stile)

2.2 Basisseite für Sommer und Winter mit Karte (Paula Spannring)

2.2.1 Index.html

Die html – Datei enthält im head – Bereich den Link zum Wiener Wappen als Shortcut 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.

2.2.2 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.

2.3 Inhalt für Winter (Paula Spannring)

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 und 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);
```

2.3.1 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 anzusehen, 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>
`;
```

2.3.2 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 durchnummertiert (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);
```

2.3.3 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(bezeichnung, 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.

2.4 Inhalt für Sommer (Maria Heinrich)

2.4.1 Allgemeines

Die Karte für Sommeraktivitäten wird mit Inhalten bestückt, die index.html-Seite ergänzt und gestyliert. Um die Seite noch etwas abzurunden, wird am Ende der Seite ein Bild eingefügt. Im Sommer eignen sich die Daten zur Lokalisierung von Plätzen und Orten mit Aktivitäten im Freien. Dazu gehören Spielplätze, Waldspielplätze, Parkanlagen, Badestellen, Grillzonen an der Donau und Fußgängerzonen in Wien zum Aufenthalt und Entspannen. Die Daten werden als *.json-Dateien zur Verfügung gestellt und unterscheiden sich inhaltlich und im Aufbau dadurch, dass sich deren types entweder um Feature-Points, oder -Polygons handelt. Feature-Points werden mit der Variablen „geoJsonPoint.“ die Feature-Polygone „feature“ aufgerufen. In der Ursprungsdatei werden die notwendigen Daten, abgespeichert unter ihren Attributen, die unter dem Punkt „properties“ in Form eines Arrays abgespeichert sind, aufgerufen.

Ein nennenswerter Unterschied beim Aufruf der Informationen besteht darin, dass beim type Point zunächst ein Popup mit den Informationen an einen Marker übergeben wird und anschließend der Marker samt Popup an die Karte gebunden wird. Beim type Polygon muss zunächst das Polygon mit einer Funktion definiert werden und ausgegeben werden, anschließend wird ein Popup definiert und an den Layer drangehangen. Dies geschieht über die Variable „layer“. Deshalb benötigt der Aufruf der Informationen aus der json-Datei eine Variable mehr. In folgender Tabelle wird verdeutlicht, wie die Informationen aus den Arrays der Ursprungsdatei aufgerufen werden (s. Tabelle 1, Abbildung 4 und Abbildung 5).

Aufruf	Pfad .properties	Attribut	Beispiel (Ausgabe)
type: Point (vgl.)			
// Badestellen		Name/Standort:	Neue Donau, Höhe Segelhafen, rechts
	Name/Standort: \${geoJsonPoint.properties.BEZEICHNUNG}	Lage:	21., Neue Donau, linkes Ufer, zwischen Brigitteiner Brücke und Brigitteiner Badebucht,
type: Polygon			
//Grillzonen			
	Lage: \${layer.feature.properties.LAGE}		

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

```
Badestellen main.js:106
▼ Object ⓘ
  ► crs: {type: 'name', properties: {...}}
  ▼ fe crs es: Array(28)
    ▼ 0:
      ► geometry: {type: 'Point', coordinates: Array(2)}
        geometry_name: "SHAPE"
        id: "BADESTELLEN0GD.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"
      WASSERTEMPEARTUR: 23.3
      WEITERE_INFO: "http://www.wien.gv.at/forschung/laboratorien/umweltmedizin/wasse
      ► [[Prototype]]: Object
      type: "Feature"
      ► [[Prototype]]: Object
    ► 1: {type: 'Feature', id: 'BADESTELLEN0GD.1582782', geometry: {...}, geometry_name: 'S...
```

Abbildung 4: Beispiel der json-Datei der Badestellen (type Point) in Wien.

Array mit 28 Features (Badestellen). geometry Point - Punktförmiger Aufruf mit „GeoJsonPoint.properties. ...“. Icon als Anzeigeeicon (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:BADE-STELEN0GD&srsName=EPSG:4326&outputFormat=json"))

```
Grillzonen: main.js:227
▼ Object ⓘ
  ► crs: {type: 'name', properties: {...}}
  ▼ features: Array(3)
    ▼ 0:
      ► geometry: {type: 'Polygon', coordinates: Array(1)}
        geometry_name: "SHAPE"
        id: "GRILLZONE0GD.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 5: Aufbau des json für Grillzonen (type Polygon). 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=ogdwien:GRILLZONE0GD&srs-Name=EPSG:4326&outputFormat=json"))

2.4.2 Point-Features

Am Beispiel der Badestellen wird der Aufbau der Inhalte der Features erläutert, die als Punkte abgespeichert sind (s. oben). Aufgerufen werden Spielplätze, Waldspielplätze, Parkanlagen und Badestellen.

Sie unterscheiden sich primär durch ihre Darstellung mittels Marker (Icons), Popup-Inhalte (s. Tabelle 2), verwendete Variablen für die URL und die URL zur Quelldatei.

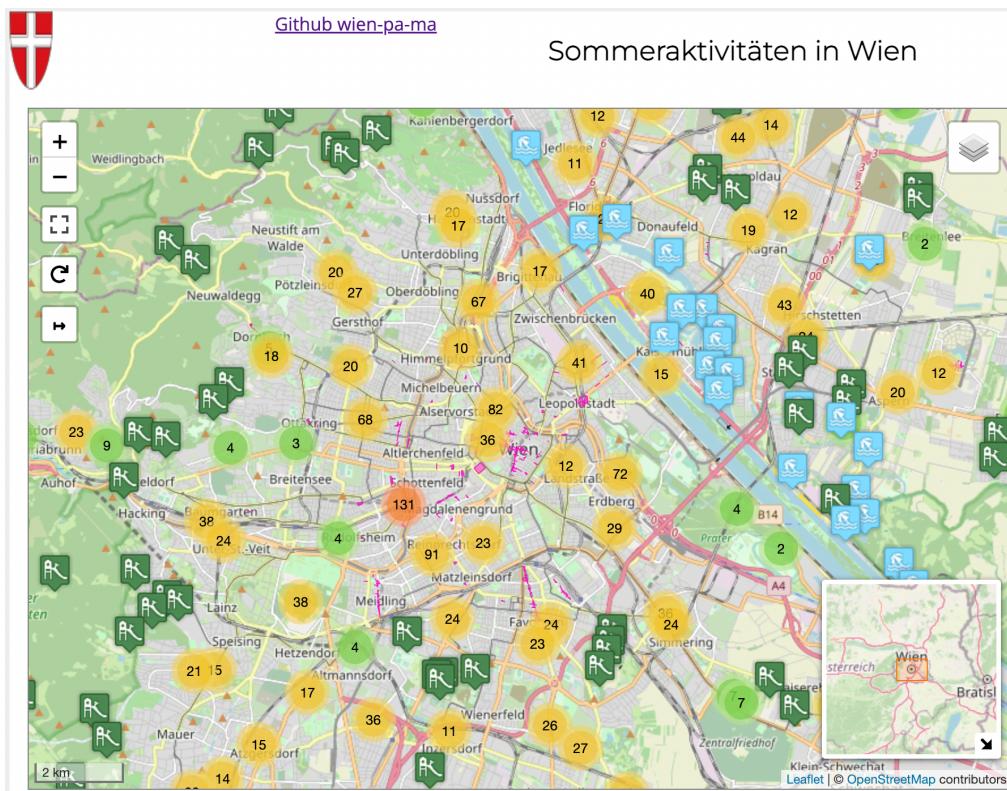


Abbildung 6: Karte der Sommerseite in Web-Ansicht. Quelle: eigene Darstellung

	Waldspielplätze	Spielplätze	Parkanlagen	Badestellen
Popup-Inhalt (Attribute)	<ul style="list-style-type: none"> • Standort • Bezirk • Angebot • Weblink 	<ul style="list-style-type: none"> • Name • Bezirk • Spielplatzelement 	<ul style="list-style-type: none"> • Name • Flächengröße • Hunde erlaubt • Spielen erlaubt • Wasser vorhanden • Öffnungszeiten • Weblink 	<ul style="list-style-type: none"> • 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.geoJSON(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 - einem Popup und Icon - 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.WASSERTEMPEARTUR}<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 – am Beispiel der Badestellen – über das Keyword „iconUrl“ mit „./icons/baden.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:BADESTELLEN0GD&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

2.4.3 Polygon-Features

Bei Flächenfeatures werden zunächst die Flächen als Layer in die Karte eingebaut und die Flächen innerhalb der Funktion mit einem Popup versehen. Dafür werden 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.

Es gibt zwei Flächenfeatures: „Fußgängerzonen“ und „Grillplätze“. Sie unterscheiden sich 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&version=1.1.0&typeName=ogdwien:FUSSGEHERZONE0GD&rsName=EPSG:4326&outputFormat=json");
```

2.4.4 Plugin „MarkerClusterGroup“

Das Plugin „MarkerClusterGroup“ wird für Spielplätze und Parkanlagen verwendet. Aufgrund der vielen Standorte und somit großen Punkte- bzw. Markerdichte wird die Karte unübersichtlich, wenn deren Inhalte aufgerufen werden. Insbesondere bei kleinem Zoom (d. h. ein größerer Kartenausschnitt), können keine aussagekräftigen Informationen entnommen werden. Deshalb sollen die Punkte zusammengefasst werden und statt aller Marker, die Anzahl der Marker in einem Label angezeigt werden (vgl. Abbildung 7). Bei kleinerem Kartenausschnitt werden weniger Marker zusammengefasst, bis schließlich bei Zoom-Stufe 17 keine Marker mehr gruppiert werden. 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
```



Abbildung 7: links: Marker ohne Gruppierung; rechts: Gruppierung und Anzeige mehrerer Marker;

3 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_schwimmbaderstandorte-wien/resource/29041605-d50d-4696-8827-bb496ac9ad75
- Silvesterpfad: <https://www.data.gv.at/katalog/dataset/f5d7a560-0599-4ee6-b9c1-5fec380b912>
- 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_fugngerzonewien/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>