# 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.

# Konzepterstellung

## 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.

## Vorgehen

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

# Erstellung der Webseiten

## 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.

|  |  |
| --- | --- |
| Ein Bild, das Text enthält.  Automatisch generierte Beschreibung  Abbildung 1: Struktur mit main.css im "Hauptordner" | Ein Bild, das Text enthält.  Automatisch generierte Beschreibung  Abbildung 2: PlugIn für Stylesheet |

## Basisseite für Sommer und Winter mit Karte

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

### 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.

### 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);

### 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>

            `;

**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);

**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.

## Inhalt für Sommer

### Main.js

#### GeoJSONS mit GeoJsonPoint

Popup

Icons suchen

Inhalte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Spielplätze | Waldspielplätze | Parkanlagen | Badestellen |
| Popup-Inhalt |  |  |  |  |
| ../icon/ | spielplatz.png | wspielplatz.png | urbanpark.png | spielplatz.png |

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.

#### GeoJson als Layer

|  |  |  |
| --- | --- | --- |
|  | Fußgängerzonen | Grillplätze |
| Popup-Inhalt |  | Lage: ${layer.feature.properties.LAGE},<br><hr>  Reservierung: ${layer.feature.properties.RESERVIERUNG},<br>  <a href="${layer.feature.properties.WEBLINK1}" target="\_blank" >Weblink</a> |
| color | #F012BE | "#FFDC00" |

### PlugIn „MarkerCluster“

Index.html

Main.js

# Quellenangaben

**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>

**Font awesome Stylesheet**

<https://cdnjs.com/>

**Daten**

Schwimmbäder: <https://www.data.gv.at/katalog/dataset/stadt-wien_schwimmbderstandortewien/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>

**Icons & Images**

<https://mapicons.mapsmarker.com/>