universität innsbruck

2024S716409 VU Geoinformatik: Web mapping

Projektbericht

zum Thema:

Erneuerbare Energien: Europa und Deutschland

Projektname: *EuropeEnergy*

Betreuung:

Klaus Förster, Bernd Öggl

Gruppenteilnehmer:

Daniel Schönung (Matrikel-Nr.: 12346290)

Niklas Schwardmann (Matrikel-Nr.: 12346123)

Marc Kaufer (Matrikel-Nr.: 01519982)

Link zu Github: https://github.com/EuropeEnergy/EuropeEnergy.github.io

Link zur Website: https://europeenergy.github.io/

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Beschreibung und Konzept

Der folgende Bericht beschreibt die Erarbeitung der Webseite “Erneuerbare Energien, Europa und Deutschland”, erarbeitet durch die Gruppe “EuropeEnergy” als Abschlussaufgabe der Lehrveranstaltung Geoinformatik: Webmapping im SS2024. Die Startseite ist unter https://europeenergy.github.io erreichbar. Von dieser sind die beiden Seiten mit der Europa- und Deutschlandkarte über eine oben positionierte Navigationsleiste erreichbar.

Bereits zu Beginn der Bearbeitungsphase bestand die Grundidee, eine Webseite zum Thema erneuerbare Energien zu erstellen, die aus einer informativen Webseite besteht, die zu zwei weiteren Webseiten mit verschiedenen Kartendarstellungen weiterleitet.

Es folgte die Suche nach passenden und nutzbaren Datensätzen, die in die Webseite implementiert werden können. Sehr schnell konnte der Eurostat-Datensatz gefunden werden und es entstand nach einigem Brainstorming die Idee für die Europakarte. Hierfür musste jedoch der thematische Datensatz mit Geodaten verknüpft werden (siehe Kapitel 4.1.). Für die zweite Karte (Deutschlandkarte) gab es verschiedene Ansätze. Der Fokus der Recherche lag hierbei allerdings auf einem Datensatz, welcher verschiedene Kraftwerkstypen in Österreich enthält. Ein solcher Datensatz war nicht auffindbar, stattdessen konnte ein Datensatz für Deutschland gefunden werden, welcher somit endgültig für die zweite Kartendarstellung verwendet wurde.

Darüber hinaus bestand zu Beginn die Überlegung eine ausführende, wissenschaftliche Recherche für die Startseite durchzuführen, auf der die aktuelle Verbreitung erneuerbarer Energien, sowie Richtlinien und Gesetze in Deutschland und Europa aufgeführt werden. Dieser Ansatz wurde wieder verworfen, da der Fokus des Projekts auf der Programmierung der Webseite liegen sollte und eine umfassende wissenschaftliche Recherche nicht der Projektzeit angemessen gewesen wäre; einige Inhalte hätten sich zudem mit den Karteninhalten überschnitten.

2. Design der drei Webseiten (Niklas)

Für die drei verschiedenen Webseiten wurden jeweils eigene html- und JavaScript-Dateien innerhalb des repositorys erstellt. Formatiert wurden alle Webseiten über dasselbe css-Dokument. Die <head>-Bereiche der jeweiligen html-Dokumente sind dabei jeweils gleich aufgebaut. Zu Beginn wird das Dokument mit <!DOCTYPE html> initialisiert, sodass Browser das Format erkennen und richtig darstellen können. Der Code <html lang=“de“> definiert es als deutschsprachiges Dokument, sodass es beispielsweise durch Google primär für Nutzer*innen in deutschsprachigen Regionen vorgeschlagen wird.

Anschließend wird mit <meta charset=“UTF-8“> die Zeichenart der Webseite definiert. <meta name="viewport" content="width=device-width, initial-scale=1.0"> sorgt dafür, dass sich die Auflösung an das von der/dem User*in genutzten Gerät anpasst. Mit dem Code <meta http-

equiv="X-UA-Compatible" content="ie=edge"> wird älteren Browsern, wie dem Internet Explorer vorgegeben, in welchem Stil die Webseite dargestellt werden soll.

Am Ende des <head>-Bereichs wird das entsprechende html-Dokument mit dem zugehörigen JavaScript-Dokument und dem allgemeinen css-Dokument verlinkt und die für die Seite verwendeten Plugins implementiert.

Der stilistische Aufbau der drei Webseiten ist grundsätzlich identisch. Oben erscheint eine Navigationsleiste, es folgt ein Bannerbild mit der Überschrift, der inhaltliche Bereich der Seite und am Ende der Seite befindet sich eine Fußleiste mit Kontakten und Informationen zum Modul.

Die Navigationsleiste – im Code als <nav> definiert – wurde in css designt und weist eine Höhe von 85 Pixeln und eine Breite von 100% auf. Durch den Befehl “position: fixed” bleibt die Navigationsleiste auch beim Scrollen am oberen Rand der Seite sichtbar. Links in der Navigationsleiste wird das selbst erstellte Logo der Organisation “EuropeEnergy” abgebildet. Ein Klick darauf leitet zur Github-Seite der Organisation weiter.

Rechtsbündig befindet sich die eigentliche Navigation in Form einer “unordered list”. Diese besteht aus den Verlinkungen Startseite, Europa, Deutschland und Impressum. Mithilfe der Funktion <a href> wurden die verschiedenen Webseiten verlinkt. Durch einen ebenfalls in css definierten Hover-Effekt färben sich die Reiter leicht grün ein. Bei einem Hover über die Startseite-Navigation wird ein Dropdown-Menü geöffnet, welches aus einer weiteren, untergeordneten “unordered list” besteht. Hier werden keine direkten links genutzt, sondern Verlinkungen innerhalb des Dokuments zu den zuvor erstellten Abschnitt-Containern, welche die einzelnen Textabschnitte umgeben. Das gleiche gilt für das Impressum. Bei einem Klick auf die Navigation springt die Website zu einer im HTML-Element Footer erstellten ID.

Es folgt ein “script” für das Auslösen der in Abbildung 1 gezeigten JavaScript-Funktion, die dazu führt, dass bei einer mobilen Ansicht die verschiedenen Reiter ausgeblendet werden und stattdessen ein sog. “Hamburger-Menü” angezeigt wird. Bei einem Klick auf das Menüsymbol erscheint ein Dropdown-Menü, welches die ausgeblendete Navigationsleiste, inklusive erweitertem Dropdown-Menü der Startseite für die mobile Ansicht anzeigt.

```
//Responsive Navigationsleiste
document.addEventListener('DOMContentLoaded', function () {
    const menuIcon = document.getElementById('menu-icon');
    const navLinks = document.getElementById('nav-links');

    menuIcon.addEventListener('click', function () {
        navLinks.classList.toggle('active');
    });
});
```

Abb. 1: JavaScript-Code der Responsiven Navigationsleiste

Unter der Navigationsleiste befindet sich die Überschrift der jeweiligen Webseite, die auf einem mithilfe von GIMP zugeschnittenen Bild angezeigt wird. Die Positionierung über dem Bild erfolgt durch die Definition einer ID für das Bild und einer darin befindlichen ID für die Überschrift. Die Überschrift wird anschließend einzeln in css positioniert, sodass sie über dem Bild zu sehen ist. Gleiches wurde mit der Bildquelle getan, um diese in ihre Position unten rechts im Bild zu bewegen.

Es folgt der eigentliche Inhalt der Webseite. Dieser unterscheidet sich auf den verschiedenen Seiten, das grundlegende Design wurde jedoch zentral in css formatiert. Dafür wurde der Hintergrund farblich an das allgemeine Farbschema angepasst. Die eigentlichen Inhalte befinden sich im <article>, welcher sich mithilfe von css-Formatierungen wie “margin”, einer anderen Hintergrundfarbe, Schatten und abgerundeten Ecken vom Hintergrund abhebt.

Am Ende der drei Seiten befindet sich der “Footer”. Dieser beinhaltet auf der linken Seite die Kontaktmöglichkeiten zu den drei Gruppenmitgliedern. Hier sind die uibk-Mailadresse, sowie das Instagram- und Github-Profil der jeweiligen Teammitglieder verlinkt. Mit der Funktion “target=“_blank“” werden die Seiten automatisch in einem neuen Tab geöffnet; durch den Tag “mailto:” vor der jeweiligen Mailadresse öffnet sich automatisch das Mailprogramm des/der Nutzer*in. In der Mitte des “footers” werden der Kurs, die Dozierenden und die Universität genannt, wobei eine direkte Verlinkung zur Startseite der Universität eingebaut ist. Auf der rechten Seite des “footers” wurde zudem das Logo der Universität eingefügt. Die Auflösung wurde ebenfalls mithilfe von GIMP reduziert und aus Formatierungsgründen in css auf eine Breite von maximal 210 Pixel beschränkt.

3. Implementierung der Startseite (Niklas)

Die Startseite des Projekts dient dazu, den Leser*innen einen Überblick über das Thema zu geben. Zunächst werden allgemeine Informationen zu erneuerbaren Energien dargestellt, anschließend werden die verschiedenen erneuerbaren Kraftwerkstypen genauer vorgestellt. Da es bei der Erstellung der Webseite primär um die Erstellung bzw. Programmierung ging, wurden die Texte von der Seite „Erneuerbare Energie Österreich“ übernommen und als direktes Zitat – inklusive Verlinkung in neuem Tab – markiert.

Da die Seite überwiegend aus Textabschnitten besteht, ist das html-Dokument “index.html” der Kern dieser Seite. Dieses Dokument wurde wie bereits beschrieben mithilfe des gemeinsamen css-Dokuments formatiert und durch eine JavaScript-Funktion erweitert.

Für jeden Abschnitt, abgesehen vom Abschnitt „Grundlagen“ wurde – wie bereits im header-Banner – ein Bild mithilfe von GIMP auf die korrekte Auflösung formatiert und anschließend zugeschnitten. Um die Überschriften innerhalb des Bilds anzuzeigen, sind die Bilder innerhalb einer ID formatiert. Innerhalb dieser ID ist wiederum die Überschrift als weitere ID formatiert, sodass die Position und der Stil der Schrift mithilfe von css angepasst werden kann.

Mithilfe von ChatGPT wurde zudem eine Funktion eingefügt, welche dazu führt, dass die Bilder, sofern auf sie gescrollt wird, von unten in die richtige Position sliden. Dies wird durch den folgenden JavaScript-Code ausgeführt:

```javascript
//Slide-In-Effekt der Bilder//
document.addEventListener('DOMContentLoaded', function () {
    const images = document.querySelectorAll('.slide-in');

    function checkVisibility() {
        images.forEach(image => {
            const rect = image.getBoundingClientRect();
            if (rect.top < window.innerHeight && rect.bottom > 0) {
                image.classList.add('visible');
            } else {
                image.classList.remove('visible');
            }
        });
    }

    window.addEventListener('scroll', checkVisibility);
    window.addEventListener('resize', checkVisibility);

    checkVisibility();
});
```

Abb. 2: JavaScript-Code für den Slide-In-Effekt der Bilder

Zunächst wird das Event des “slide-in” erstellt. Es folgt eine Funktion, welche für jedes einzelne Element definiert, ob es beim Start der Seite sichtbar ist. Mithilfe der if-Funktion werden die Elemente mit der ID „image“, die im index.html als solche definiert wurde, sichtbar dargestellt. Für die restlichen Elemente wird dieser Effekt entfernt. Mithilfe von

“window.addEventListener('scroll'/resize, checkVisibility)” wird dieser Effekt bei jedem “scroll” und der Änderung der Fenstergröße durch den User wiederholt, sodass der Effekt mehrfach auftreten kann und die Bilder wieder verschwinden, wenn von ihnen weggescrollt wird. Im css-Dokument wurde eingestellt, das die “opacity” des Elements, wenn sie nicht sichtbar ist, 0 beträgt und sich nach dem “scroll” auf das Element innerhalb von 0,5 Sekunden zu einer “opacity” von 1 wandelt. Zudem wurde definiert, dass der “slide-in” von unten auf die richtige Position durchgeführt werden soll.

4. Implementierung der Europa-Karte (Daniel)

4.1. Datenaufbereitung

Um das Ziel der Erstellung einer Thematischen Karte für erneuerbare Energien in Europa zu erreichen, benötigt man idealerweise ein GeoJSON-File, welches sowohl Polygondaten zu den europäischen Ländern als auch thematische Daten zu den erneuerbaren Energien enthält, um die Geometriedaten anschließend über die Properties mit thematischen Eigenschaften zu versehen. Leider wurde ein solches File bei der Recherche nicht gefunden. Eine Verknüpfung von zwei verschiedenen Files mit Geometriedaten und Thematischen Daten innerhalb von JavaScript erschien zudem nach einiger Recherchearbeit aussichtslos. Aus diesem Grund wurden von EUROSTAT zunächst rein thematische Daten („Anteil erneuerbarer Energiequellen am gesamten Bruttoendenergieverbrauch“) als CSV-Datei exportiert. Dieser Datensatz erschien besonders geeignet, da dieser neben dem allgemeinen Anteil erneuerbarer Energien am gesamten Bruttoendenergieverbrauch unter anderem einzelne erneuerbare Energieformen (z.B. Wasserkraft) für jedes Land beinhaltet.

Des Weiteren wurde von dem “Open Data Portal Rhein-Kreis-Neus" Geometriedaten zu den Ländern (Polygondaten) als CSV-Datei heruntergeladen. Das CSV-Dateiformat ermöglichte es nun, innerhalb von EXCEL die Länderpolygondaten mit den thematischen Daten zu verknüpfen bzw. in eine gemeinsame Tabelle zu kopieren. Anschließend wurde mithilfe eines Online-Converters die erstellte CSV-Datei zu einer GeoJSON Datei konvertiert (https://www.fileconverts.com/csv/csv-to-geojson/). Schließlich wurden innerhalb des GeoJSON noch einzelne Fehler korrigiert, um die Datei für JavaScript lesbar zu machen.

4.2. Implementierung

Grundsätzlich wurden folgende Hilfestellungen/Tutorials für die Implementierung der Europa-Karte verwendet:

- Das Grundlegende Wissen der Lehrveranstaltung zu Erstellung einer Leaflet-Karte, Einbindung des GeoJSONS, Erstellung von PopUps, Referenzierung der Properties etc.
- Leaflet Tutorial „Interactive Choropleth Map“ als Basis für die Thematische Karte (https://leafletjs.com/examples/choropleth/)
- Codebeispiele der Google-Charts Diagramme (Beispiel: https://developers.google.com/chart/interactive/docs/gallery/piechart?hl=de#donut)
- Codebeispiele für die Einbindung von Plugins (z.B. Erstellung einer Sidebar)
- Eigene Erfahrung aus dem Kartographie Bachelor-Studium, YouTube Tutorials, Recherche oder ChatGBT für vereinzelte Fragen zu Plugins (Es wurde kein Code kopiert; das Codieren, Zusammenführen der Elemente sowie die Anwendung auf den eigenen Datensatz erfolgte durchweg eigenständig)

4.2.1. Erstellung eines Leaflet GeoJSON-Objektes

```
//DATENIMPORT GEOJSON EUROPA-DATEN


async function showGeojsonEU(url) {
    let response = await fetch(url);
    let geojson = await response.json();


//ERSTELLUNG LEAFLET GEOJSON OBJEKT


    L.geoJSON(geojson, {
        style: style,
        onEachFeature: onEachFeature
    }).addTo(mapeu)
};
```

Abb. 3: Einbindung der Lokalen GeoJSON-Datei (Europa-Karte)

Neben der Erstellung einer Leaflet-Karte, der Einbindung von Hintergrundlayern und der Darstellung einer Maßstabsleiste wurde zunächst, wie auch in der Lehrveranstaltung in vorausgegangenen Projekten mithilfe einer async-Funktion das GeoJSON-File in die Anwendung innerhalb einer Variable “geojson” eingebunden. Einziger Unterschied zur Lehrveranstaltung ist hierbei, dass die URL-Referenz, welche beim Funktionsaufruf an die Funktion übergeben wird, auf eine lokale Datei (das erstellte GeoJSON-Europe-File) statt auf einen Server verweist. Es wurde ein Leaflet-GeoJSON Objekt erzeugt, welches die Daten des GeoJSONs auf der Europa-Karte (addTo(mapeu)) visualisieren soll. Hierfür wurden für die Optionen “Style” und “OnEachFeature” jeweils zwei eigenständige gleichnamige Funktionen erstellt, auf die hierbei verwiesen wird, um eine verbesserte Struktur zu erreichen.

4.2.2. Style-Funktion

```
//STYLE-Funktion GEOJSON-Objekt (Einfärbung der einzelnen Länderpolygone)

function style(feature) {
    if (feature.properties.Renewables_and_biofuels == null) {
        return {
            fillPattern: stripePattern,
            weight: 2,
            color: "white"
        };
    }

    else {
        return {
            fillColor: getColor(parseInt(feature.properties.Renewables_and_biofuels)),  //Hier ParseInt da Zahlenwert in JSON als String gespeichert
            weight: 2,
            opacity: 1,
            color: "white",
            fillOpacity: 1
        };
    }
}
```

Abb. 4: Style-Funktion mit If/Else-Anweisung (Europa-Karte)

Die Funktion “Style” liest die einzelnen Features des GeoJSONs ein. Zunächst wurde innerhalb der Funktion eine IF/ELSE-Anweisung erstellt, da für die verschiedenen Länder je nach Datenverfügbarkeit eine unterschiedliche Visualisierung erfolgen soll.

IF-Anweisung: Handelt es sich um ein Feature (Europäisches Land), welches keine EUROSTAT-Daten zu erneuerbaren Energien beinhaltet, soll ein gestreiftes Muster statt einer Farbe das Polygon ausfüllen, um kartographisch eine Fehlinterpretation zu vermeiden. Hierbei kommt das Plugin „Leaflet Pattern“ zum Einsatz. Das Plugin ermöglicht die Style-Option „fillPattern”. Es wird auf ein zuvor Erstelltes Leaflet.Pattern-Objekt verwiesen, bei dem es sich in diesem Fall um ein „StripePattern“-Objekt handelt, um ein gestreiftes Muster zu kreieren. Hierbei werden einige Styling-Optionen wie Farbe, Winkel, Strichbreite etc. angegeben. Das StripePattern Objekt wird zudem der Leaflet Europa-Karte (mapeu) hinzugefügt.

ELSE-Anweisung: Handelt es sich um ein Feature (Europäisches Land), welches EUROSTAT-Daten zu erneuerbaren Energien beinhaltet, wird über die Style-Option „fillColor“ auf eine neu erstellte Funktion “getColor” verwiesen. Übergeben wird hierbei der Feature-Wert „Anteil erneuerbarer Energiequellen am gesamten Bruttoendenergieverbrauch“ welcher in den GeoJSON-Properties unter „Renewables_and_biofuels“ gespeichert ist. Der Wert ist innerhalb des GeoJSON als String gespeichert, womit dieser zuvor noch mithilfe der Funktion „parseInt“ in einen Integer Datentyp umgewandelt wird.

Die “getColor” Funktion überprüft nun für jedes Feature den übergebenen Integer Wert aus der Style-Funktion und gibt einen entsprechenden Farbwert (je nach Klasseneinordnung) über “return” zurück. Hierfür wurden zunächst innerhalb eines Arrays „klassen“ die einzelnen Klassengrenzen definiert, welche dann in der “getColor” Funktion als Klassengrenzen referenziert werden. Möchte man andere Klassengrenzen definieren, kann man dies zentral in diesem Array tun. Die sequenziellen Farbabstufungen (einzelnen Farbwerte im HEX-Code) wurden dabei eigenständig mithilfe des Farbraums innerhalb der CSS-Datei erstellt und orientieren sich an dem

gesamten Farbschema der Website. Je dunkler der Farbwert, desto höher ist der Anteil an erneuerbaren Energiequellen in dem entsprechenden Land.

4.2.3. Legende der Karte

Damit die Karte auch eine Legende erhält, wird hierfür ein neues L.control-Element erstellt. Innerhalb einer DIV-Variable wird hierbei der Titel der Legende gespeichert und mithilfe von HTML-Anweisungen formatiert. Mit einer FOR-Schleife werden anschließend die einzelnen Legendenelemente erstellt. Hierfür wird erneut der zuvor erstelle Array „klassen“ benötigt, der zum einen vorgibt, wie lange die FOR-Schleife durchläuft und zum anderen die Farben aus der getColor Funktion abruft sowie die einzelnen Klassenbeschriftungen erstellt, welche untereinander aufgeführt werden.

4.2.4. OnEachFeature-Funktion

Neben der Style-Funktion wird bei Erstellung des Leaflet-GeoJSON-Objektes zudem auf eine “onEachFeature”-Funktion verwiesen. Hierbei soll nun programmiert werden, was passiert, wenn einzelne Länder angeklickt werden. Mit einer IF-Anweisung wird zunächst wieder abgefragt, ob das angeklickte Feature keine Daten zu den erneuerbaren Energien besitzt. Ist dies der Fall, wird ein einfaches Pop-Up erstellt, welches den Namen des Landes (Über Properties referenziert) und die Meldung „Leider keine EUROSTAT-Daten verfügbar“ beinhaltet. Besitzt das angeklickte Feature Daten zu erneuerbaren Energien, wird mithilfe von “layer.on (click: ClickOnFeature)” ein Click-Event durchgeführt, bei dem auf die Funktion ClickOnFeature verwiesen wird.

4.2.5. ClickOnFeature-Funktion

```
//Funktion ClickOnFeature für einzelne Click Events für die Diagrammerstellung
function ClickOnFeature(e) {

    if (e.target.feature.properties.Renewables_and_biofuels == null) {
        sidebar.hide()
    }

    else {
        //Öffnen der Sidebar und Definition des Inhalts
        sidebar.setContent(`<button id="b1"><i class="fa-regular fa-circle-xmark" font-size="50px"></i></button> <br> <h1>${e.target.feature.properties.preferred_term}
        (${(parseFloat((e.target.feature.properties.Renewables_and_biofuels).replace(',', '.'))).toFixed(1)} %)</h1><br>
        <hr class="Strich_Sidebar"><p><p style="font-size:18px;">Energieform (Anteile in %)
        </h2><div id="Diagramm"></div><br><hr><br></p><p><div id="Tabelle"></div></p><p style="font-size: 12px;">*Anteil
        der Kategorie am gesamten Bruttoendenergieverbrauch</p> <br><br> <p><b>Quelle: </b><i>EUROSTAT (Stand 2021)</i><br>
        <a href="https://ec.europa.eu/eurostat/databrowser/view/nrg_ind_rftce/default/table?lang=en&category=nrg.nrg_quant.nrg_quanta.nrg_ind_share" target="_blanc">Link zum Datensatz</a></p>`).show();

        //Erzeugung des Buttons zum Schließen
        document.getElementById('b1').addEventListener('click', function () {
            sidebar.hide();
        })


        //Variablen um Werte aus GeoJSON abzugreifen (bzw. aus dem Feature, welches angeklickt wurde) - da String, Umwandlung in Nummer notwendig!
        let Biomasse = parseFloat((e.target.feature.properties.Sustainable_primary_solid_biofuels).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Charcoal).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Sustainable_biofuels).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Sustainable_bioliquids).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Sustainable_biogases).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Renewable_municipal_waste).replace(',', '.'));

        let Wasserkraft = parseFloat((e.target.feature.properties.Hydro).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Tide_wave_ocean).replace(',', '.'));

        let Wind = parseFloat((e.target.feature.properties.Wind).replace(',', '.'));
        let Geothermie = parseFloat((e.target.feature.properties.Geothermal).replace(',', '.'));

        let Sonnenenergie = parseFloat((e.target.feature.properties.Solar_thermal).replace(',', '.'))
            + parseFloat((e.target.feature.properties.Solar_photovoltaic).replace(',', '.'));

        let Wärmepumpen = parseFloat((e.target.feature.properties.Ambient_heat_heat_pumps).replace(',', '.'))
        let erneuerbare_Kuehlung = parseFloat((e.target.feature.properties.Renewable_cooling).replace(',', '.'))
```

Abb. 5: ClickOnFeature Funktion – Teil 1 (Europa-Karte)

```
        //Anlegen des Arrays, welcher dann für die Diagrammerstellung notwendig ist und an drawChart übergeben wird
        let tabellenbezeichnung = ['Biomasse', 'Wasserkraft', 'Wind', 'Geothermie', 'Solarenergie', 'Wärmepumpen', 'erneuerbare Kühlung']
        let tabellenwerte = [parseFloat(Biomasse.toFixed(1)), parseFloat(Wasserkraft.toFixed(1)), parseFloat(Wind.toFixed(1)),
        parseFloat(Geothermie.toFixed(1)), parseFloat(Sonnenenergie.toFixed(1)), parseFloat(Wärmepumpen.toFixed(1)), parseFloat(erneuerbare_Kuehlung.toFixed(1))]

        let diagrammdaten = []

        for (let a = 0; a < tabellenwerte.length; a++) {
            diagrammdaten[a] = [tabellenbezeichnung[a], tabellenwerte[a]]
        }

        //Erstellung der einzelnen Diagramme

        //PIECHART
        google.charts.load('current', { packages: ['corechart'] });
        google.charts.setOnLoadCallback(function () {
            drawChart(diagrammdaten)
        });

        //TABELLE
        google.charts.load('current', { 'packages': ['table'] });
        google.charts.setOnLoadCallback(function () {
            drawTable(diagrammdaten);

        });
    }
}
```

Abb. 6: ClickOnFeature Funktion – Teil 2 (Europa-Karte)

Innerhalb der Funktion "*ClickOnFeature*" wird nun das angeklickte Feature (in diesem Fall im Parameter „e") behandelt. Infolgedessen können die Properties-Inhalte der GeoJSON-Datei nun mehr über *e.target* referenziert werden. Eine IF-/ELSE Anweisung ermöglicht zudem das Schließen einer potenziell zuvor geöffneten Sidebar, falls auf ein europäisches Land ohne Daten geklickt wurde. Wird auf ein Land geklickt, welches im Datensatz thematische Daten enthält, wird für das angeklickte Feature eine neue Sidebar erstellt, bzw. ein neuer Inhalt definiert.

Ein Leaflet-Sidebar-Objekt wurde zuvor schon erstellt (entsprechend des Plugins *leaflet-sidebar*) und der Europakarte hinzugefügt. In dieser "*ClickOnFeature*"-Funktion wird nun der "Sidebarcontent" für das angeklickte Feature definiert. Es wird hierbei eine eigene HTML-Struktur mit einzelnen Überschriften, Textblöcken und DIV-Elementen angelegt. Da der Button, mit dem

man die Sidebar schließen kann, leider nicht erscheint (ggf. Fehler in Plugin?), wurde ein eigener Button „b1“ mithilfe eines “*EventListeners”* erstellt, der bei einem Klick die Sidebar schließt (*sidebar.hide*()). Für den Button wurde ein Icon von “*FontAwesome”* (https://fontawesome.com/) verwendet.

Die Sidebar enthält zudem zwei spezifische Google-Chart-Diagramme („Google Pie-Chart“ und „Google-Table“) welche in zwei verschiedenen DIV-Elementen untereinander angezeigt werden. Um diese Diagramme zu erstellen, werden zunächst einige Daten sortiert und in einem zentralen Array („diagrammdaten“) abgespeichert, welcher dem vorgegebenen Format für die Erstellung von Google-Chart Diagrammen entspricht. Zunächst werden die Daten einzelner Unterkategorien (Wie zum Bespiel Windkraft an Land und Windkraft auf dem Meer) zu einheitlichen Gesamtkategorien (Windkraft) addiert, sodass am Ende folgende erneuerbare allgemeine Energieformen verbleiben: Biomasse, Wasserkraft, Solarenergie, Windkraft, Wärmepumpen, erneuerbare Kühlung. Hierbei werden die im String-Format verfügbaren Daten in Dezimalzahlen umgewandelt und auf eine Nachkommastelle gerundet (folgende Methoden waren hierfür notwendig: “replace”, “parseFloat”, “toFixed”). Mit einer FOR-Schleife wird anschließend der Array „diagrammdaten“ befüllt, womit jeder Wert eine Bezeichnung (z.B. „Biomasse“) und einen entsprechenden Datensatz (z.B. 20.6) erhält. Der Array wird anschließend an die Funktionen “drawChart” und “drawTable” übergeben und bildet die Grundlage für die Diagramm-Visualisierung der Google-Chart Diagramme.

4.2.6. Google-Charts Diagramme

```
//Funktion für Erstellung eines Google-Chart Diagramms

function drawChart(diagrammdaten) {
    let data = new google.visualization.DataTable();
    data.addColumn('string', 'Energietyp')
    data.addColumn('number', 'Prozentzahl')
    data.addRows(diagrammdaten)

    var options = {
        pieHole: 0.4,
        slices: { 0: { color: '#b0a48e' }, 1: { color: '#8AA2D1' }, 2: { color: '#65C8CF' }, 3: { color: '#C59E74' }, 4: { color: '#D0D07B' }, 5: { color: '#DE7080' }, 6: { color: '#B374CA' } },
        backgroundColor: 'white',
        width: "60%",
        height: "60%",
        legend: { position: 'bottom' },
        pieSliceTextStyle: {
            color: 'black',
        }

    };

    var chart = new google.visualization.PieChart(document.getElementById('Diagramm'));
    chart.draw(data, options);
}
```

Abb. 7: Beispielhafte Funktion der Erstellung eines Google-Chart Diagramms (Europa-Karte)

In diesem Fall die Methode “drawChart” aufgeführt, mit der das „Pie-Chart“ Diagramm erstellt wird. Hier wird zunächst eine „DataTable“ mit zwei Spalten (Spalte 1: Energietyp (String); Spalte 2: Prozentzahl (Number)) erstellt. Der zuvor befüllte und übergebene Array mit den einzelnen Datensätzen (diagrammdaten) füllt anschließend die Zeilen der Tabelle. Innerhalb der Variable Options werden zudem noch die Farben für die einzelne Unterkategorien (”slices”) angegeben sowie weitere Styling-Parameter. Das Chart wird anschließend innerhalb des DIV-Elements in der Sidebar kreiert. Die Erstellung der Google-Charts Tabelle erfolgt analog dazu.

5. Implementierung der Deutschland-Karte (Marc)

Die dritte Seite des Projekts *Europe Energy* beinhaltet eine Karte von Deutschland, welche ähnlich die der Europaseite, einen Überblick über verschiedene Aspekte erneuerbarer Energie liefern soll. Hierbei werden vier Formen zur Gewinnung erneuerbarer Energie (Wind, Wasser, Photovoltaik und Biomasse) anhand der entsprechenden Anlagen und Kraftwerkstypen dargestellt. Diese werden über separat anwählbare Layer sichtbar, welche dann wiederrum über entsprechende Pop-ups weitere Informationen über das jeweilige Kraftwerk oder die Anlage liefern. Weiters kann über eine Suchleiste aus den Landkreisen in Deutschland gewählt werden, wodurch eine spezifischere Standortwahl und Informationssuche möglich ist. Im Folgenden wird zunächst erläutert, welche Daten zur Veranschaulichung gebracht und wie diese, zur besseren Handhabung, vor der Implementierung bearbeitet wurden. Danach wird anhand des zugrundeliegenden *java-Scripts* schrittweise dargestellt, wie die Daten implementiert wurden und welche weiteren Schritte für deren Darstellung nötig waren.

5.1. Datenaufbereitung

Für die Implementierung der genannten Anlagen zur Energieproduktion in Deutschland standen äußerst umfangreiche Datensätze (*geoJSON*) von Manske, Grosch und Schmied (2022) zu Verfügung. Diese beinhalten die entsprechenden Standorte (Koordinaten) von Windkraft- (On- und Offshore), Biogas- (auch *liquid fuel*) und Wasserkraftanlagen, sowie von größeren Freiflächen und Agri-Photovoltaik-Anlagen, welche allerdings als Polygone hinterlegt sind. Neben den Koordinaten lassen sich aus den Datensätzen noch zahlreiche weitere Informationen über die Anlagen abrufen, unter anderem das Datum der Inbetriebnahme, die installierte Kapazität und eine spezifischere Unterscheidung des Anlagentyps. Aufgrund des enormen Umfangs der einzelnen Datensätze, mussten diese vor der Implementierung in die Karte doch deutlich verkleinert werden, um lange Ladezeiten zu vermeiden. Hierfür wurden die *geoJSON files* in QGIS geladen und mittels Editierfunktion bearbeitet. Hierbei wurden dann ganze Spalten, welche für die weitere Darstellung nicht benötigt wurden gelöscht. Weiters wurden auch alle Anlagen und Kraftwerke entfernt, welche dem ebenfalls enthaltenen Datum der Abschaltung zufolge, mittlerweile bereits außer Betrieb waren. Die aus diesem Schritt hervorgegangenen und deutlich verkleinerten *files* wurden für die weitere Bearbeitung im Projektordner gespeichert.

Der zweite Datensatz, welcher die genannten und über eine Suchleiste auswählbaren Landkreise Deutschlands enthält, konnte über die Website *opendatalab.de* geladen werden. Hier konnte bereits vor dem Download Umfang (z.B. Gemeinden oder Landkreise) und Genauigkeit (hoch, mittel und stark vereinfacht) des Datensatzes gewählt werden, wodurch vor der Implementierung keine weitere Bearbeitung mehr nötig war.

5.2. Implementierung

Wie bereits beschrieben, wird die Initialisierung der Karte, sowie die weitere Implementierung der beschriebenen Daten, schrittweise anhand des *java-scripts deutschland.js* erläutert.

Initialisierung der Karte

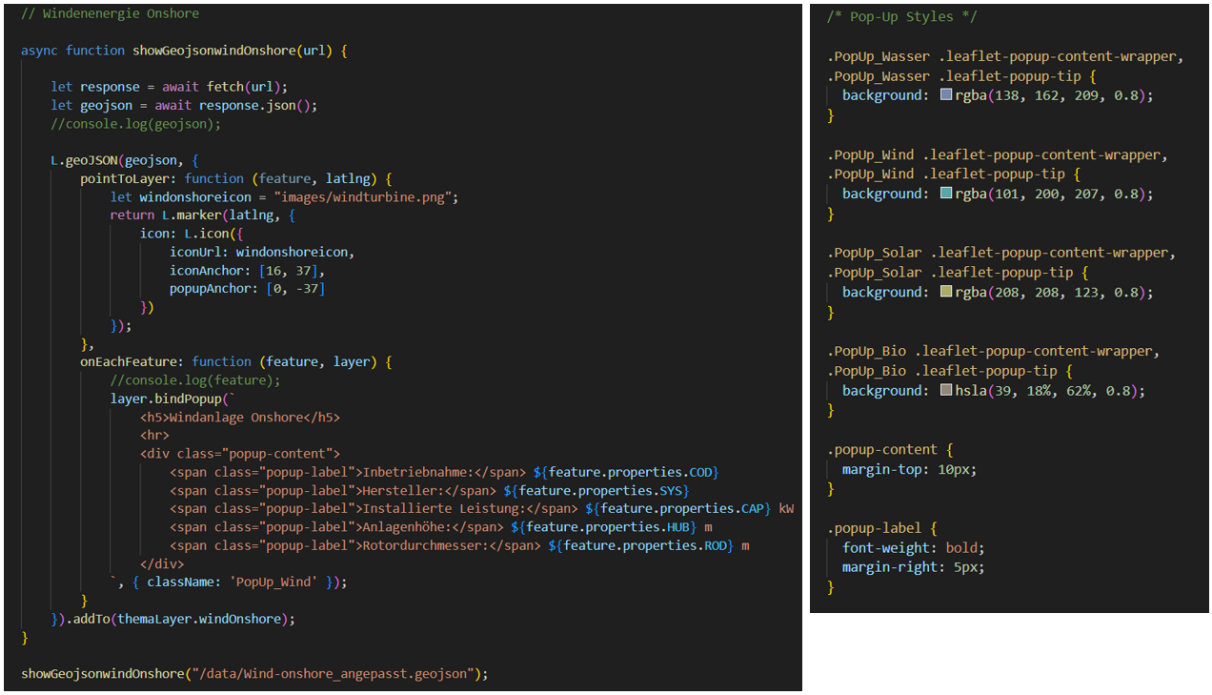
Im ersten Schritt wurde die *Leaflet-Map* initialisiert und an den Koordinaten des „offiziellen“ Mittelpunkts Deutschlands in der Nähe der Stadt Besse (Wikipedia 2024) ausgerichtet. In weiterer Folge wurde das *Leaflet Plugin Fullscreen*, der Maßstab und die Zoom Kontrolle implementiert. Für die *Layer Control* wurden über das *Leaflet Provider Plugin* drei Hintergrundkarten geladen und im gleichen Schritt die Auswahl der Thema Layer (Wind On- und Offshore, Solar, Wasserkraft und Biomasse) definiert. Aufgrund der Vielzahl der einzelnen Anlagen wurden diese der Übersichtlichkeit und des einfacheren Handlings halber, mithilfe des entsprechenden *Leaflet-Plugins geclustert* (siehe Abbildung 8). In einer späteren Phase der Arbeit, konnten diesen Clustern dann eine eigens definierte Farbe je Energieform zugewiesen werden.

```
// Layerauswahl Energietyp
let themaLayer = {
    solar: L.featureGroup(),
    windOnshore: L.markerClusterGroup({
        disableClusteringAtZoom: 17,
        iconCreateFunction: function (cluster) {
            return L.divIcon({
                html: `<div style="background-color: #65C8CF; width: 30px; height: 30px; border-radius: 50%; display: flex; align-items:
                center; justify-content: center;">${cluster.getChildCount()}</div>`,
                className: 'custom-cluster-icon'
            });
        }
    }),
    windOffshore: L.markerClusterGroup({
        disableClusteringAtZoom: 17,
        iconCreateFunction: function (cluster) {
            return L.divIcon({
                html: `<div style="background-color: #65C8CF; width: 30px; height: 30px; border-radius: 50%; display: flex; align-items:
                center; justify-content: center;">${cluster.getChildCount()}</div>`,
                className: 'custom-cluster-icon'
            });
```

Abb. 8: Funktion zur Erstellung individueller Cluster (Deutschland-Karte)

Import der GeoJSON Anlagen-Daten

Der Import der beschriebenen *geoJSON files* erfolgte, wie bei der Europa-Karte und aufgrund der größeren Datenmenge, auch in diesem Fall mithilfe einer *async function* (siehe Abbildung 9). Hier wurde dann zum einen der jeweilige Style des Layers (nur für die PV-Anlagen: Farbe und Transparenz), bzw. ein spezifisches Icon bestimmt und zum anderen individuelle Pop-ups definiert, welche bei der Auswahl einzelner *features* zusätzliche Informationen liefern können. Wie bereits beschrieben, wurden die einzelnen *geoJSON files* nach der Bearbeitung lokal gespeichert und daher von der Funktion über einen relativen Pfad zum entsprechenden Ordner abgerufen. Gleiches gilt für die verwendeten Icons, welche über die Website *mapicons.mapsmarker.com* gefunden und in der entsprechenden Farbe generiert wurden.



```
// Windenenergie Onshore

async function showGeojsonwindOnshore(url) {

    let response = await fetch(url);
    let geojson = await response.json();
    //console.log(geojson);

    L.geoJSON(geojson, {
        pointToLayer: function (feature, latlng) {
            let windonshoreicon = "images/windturbine.png";
            return L.marker(latlng, {
                icon: L.icon({
                    iconUrl: windonshoreicon,
                    iconAnchor: [16, 37],
                    popupAnchor: [0, -37]
                })
            });
        },
        onEachFeature: function (feature, layer) {
            //console.log(feature);
            layer.bindPopup(`
                <h5>Windanlage Onshore</h5>
                <hr>
                <div class="popup-content">
                    <span class="popup-label">Inbetriebnahme:</span> ${feature.properties.COD}
                    <span class="popup-label">Hersteller:</span> ${feature.properties.SYS}
                    <span class="popup-label">Installierte Leistung:</span> ${feature.properties.CAP} kW
                    <span class="popup-label">Anlagenhöhe:</span> ${feature.properties.HUB} m
                    <span class="popup-label">Rotordurchmesser:</span> ${feature.properties.ROD} m
                </div>
            `, { className: 'PopUp_Wind' });
        }
    }).addTo(themaLayer.windOnshore);
}

showGeojsonwindOnshore("/data/Wind-onshore_angepasst.geojson");
```

```
/* Pop-Up Styles */

.PopUp_Wasser .leaflet-popup-content-wrapper,
.PopUp_Wasser .leaflet-popup-tip {
  background: rgba(138, 162, 209, 0.8);
}

.PopUp_Wind .leaflet-popup-content-wrapper,
.PopUp_Wind .leaflet-popup-tip {
  background: rgba(101, 200, 207, 0.8);
}

.PopUp_Solar .leaflet-popup-content-wrapper,
.PopUp_Solar .leaflet-popup-tip {
  background: rgba(208, 208, 123, 0.8);
}

.PopUp_Bio .leaflet-popup-content-wrapper,
.PopUp_Bio .leaflet-popup-tip {
  background: hsla(39, 18%, 62%, 0.8);
}

.popup-content {
  margin-top: 10px;
}

.popup-label {
  font-weight: bold;
  margin-right: 5px;
}
```

Abb. 9: links: Implementierung der geoJSON-Daten; rechts: Style der individuellen Pop-ups im main.css (Deutschland-Karte)

Hinsichtlich der Pop-ups ist noch hinzuzufügen, dass diese im weiteren Verlauf der Arbeit ebenfalls eine spezifische Färbung erhalten sollten. Dies gelang dann nach unterschiedlichen Lösungsversuchen, über den *leaflet-popup-content-wrapper* und die Definition mehrerer „Style-Klassen“, welche dadurch im zugehörigen *main.css file* bearbeitet werden konnten (siehe Abbildung 9). Auf diese Weise konnten die Pop-ups für jeden Anlagentyp individuell eingefärbt und leicht transparent dargestellt werden. Das endgültige Ergebnis ist hierbei eine Kombination aus Beispielen aus der VU-Workload und weiteren, mithilfe von ChatGPT recherchierten Lösungswegen.

Import der Landkreise Deutschlands

Im nächsten Schritt wurden die bereits genannten Landkreise Deutschlands von der Webseite opendatalab.de geladen und integriert. Hier bestand ebenfalls die Möglichkeit, einen noch deutlich umfangreicheren Datensatz der Gemeinden zu wählen, welcher allerdings unter Umständen für deutlich längere Ladezeiten gesorgt hätte und für den erdachten Zweck der Deutschland-Karte nicht notwendig war. Aus diesen Gründen wurde auch unter der hier als *Simplify* bezeichneten Download-Auswahl die Option „200 – stark vereinfacht“ gewählt. Die bereits als *geoJson file* verfügbaren Landkreise, wurden in der Folge lokal im entsprechenden Data-Ordner des Projekts gespeichert und wie die Anlagen zur Energieerzeugung über eine *async-function* in die Karte eingebunden (siehe Abbildung 10). Hier bestand dann die erste Aufgabe darin, den Style des Layers so anzupassen, dass dieser beim Aufruf der html-Seite nicht unmittelbar sichtbar war. Im nächsten Schritt wurde dann eine Suchleiste implementiert, mithilfe derer ein gewünschter Landkreis gewählt werden kann und in der Folge durch eine farbigen Rahmenlinie auf der Karte hervorgehoben wird. Dies gelang mithilfe des *Leaflet-Plug-ins*

```
// Landkreise integrieren und Suchfunktion

async function showGeojsonLandkreise(url) {
    let response = await fetch(url);
    let geojson = await response.json();
    console.log(geojson);

    let landkreise = L.geoJSON(geojson, {
        style: function (feature) {
            return {
                color: 'transparent',
                fillColor: 'transparent',
                weight: 2,
                opacity: 0,
                fillOpacity: 0
            };
        }
    });
```

Abb. 10: Implementieren der geoJSON-Landkreise per async-function – Teil 1 (Deutschland-Karte)

„Control-Search“, welches im zweiten Abschnitt der *async-function* miteingebunden wurde (siehe Abbildung 11). Die entsprechende Layer-Control hierfür bestand dann wiederum aus zwei Abschnitten: Im ersten wurde das entsprechende Suchfeld initialisiert mit entsprechenden Eigenschaften versehen. In diesem Fall sollte das Feld darauf hinweisen, dass durch Eingabe aus den Landkreisen Deutschlands gewählt werden kann. In der Beschreibung des Plug-ins ist das Suchfeld mit einem Lupen-Icon am rechten Rand ausgestattet, mit welchem das Feld geöffnet und geschossen werden kann. Leider war es bis zuletzt nicht möglich, diese Funktion auch im vorliegenden Projekt zu implementieren. Im zweiten Abschnitt der Layer-Control wurde dann noch der Style des angezeigten *features* definiert, welcher nach Auswahl im Suchfeld, in einem dann leicht vergrößerten Ausschnitt der Karte hervorgehoben wird.

```
    let searchControl = new L.Control.Search({
        markerLocation: true,
        layer: landkreise,
        propertyName: 'GEN',
        marker: false,
        textPlaceholder: 'Such dir deinen Landkreis',
        textErr: 'Versuchs nochmal',
        collapsed: false,
        position: 'topleft',
        moveToLocation: function (latlng, title, map) {
            var zoom = map.getBoundsZoom(latlng.layer.getBounds());
            map.setView(latlng, zoom);
        }
    });

    searchControl.on('search:locationfound', function (e) {
        // Stil des gefundenen Features anpassen
        e.layer.setStyle({ fillColor: 'transparent', color: '#3abfe8', opacity: 0.6, fillOpacity: 0.6 });
    });

    mapde.addControl(searchControl);
};

showGeojsonLandkreise("/data/landkreise.geojson");
```

Abb. 11: Layer-Control des Plug-ins „search-control“ in der async-function - Teil 2 (Deutschland-Karte)

Herausforderungen und Probleme

Das erste Problem der Aufgabe war die Suche nach passenden Datensätzen im richtigen Format. Insbesondere der Datensatz zur Deutschlandkarte war schwierig zu finden. Ein vergleichbarer Datensatz für Österreich konnte nicht gefunden werden. Zudem war der Datensatz mit den verschiedenen Kraftwerkstypen in Deutschland zu umfangreich für das Projekt. Der Datensatz wurde bereits gekürzt, bietet jedoch nach wie vor zu viele Informationen, sodass die implementierten *Cluster* teilweise zu viele Kraftwerke abdecken. Dennoch wird hierbei Vollständigkeit garantiert, was vor allem bei der spezifischen Suche nach einem Landkreis zum Vorteil wird.

Sehr zeitaufwendig war auch die vorbereitende Erstellung des Datensatzes für erneuerbare Energien in Europa. Hierfür war eine Verknüpfung von thematischen Daten (EUROSTAT) und Geodaten (Länderpolygone) erforderlich, was schließlich über die Zusammenführung zweier CSS-Dateien und einer anschließenden Konvertierung in ein *GeoJSON* gelöst wurde. Im Idealfall bindet man Datensätze über einen Server-Link in die Anwendung ein (beispielsweise auch möglich über EUROSTAT), um auch eine beständige Aktualität zu bewahren. Leider war dies aufgrund der fehlenden Geometriedaten in diesem Falle nicht möglich.

In der Deutschlandkarte bot die Suchleiste einige Schwierigkeiten, da sie sich nicht wie ursprünglich geplant direkt Oberhalb der Zoom- und Fullscreen-Funktion anordnen ließ, sodass die anderen Symbole nach unten links in der Karte verschoben werden mussten. Auch die Anwendung des Plugins „Sidebar" in der Europakarte bat einige Herausforderung, insbesondere die Erstellung eines eigenen Buttons zum Schließen sowie eine ideale Anordnung der Elemente innerhalb der Sidebar. Es bleibt zudem zu erwähnen, dass die Europakarte zwar einen guten Überblick bietet, jedoch sowohl kartographisch als auch inhaltlich noch ausbaufähig ist. Dies entspricht vor Allem die Farbwahl, die Darstellung der Legende, die Definition der Klassengrenzen.

Gegen Ende der Bearbeitungszeit ist zudem aufgefallen, dass die Webseite in der mobilen Ansicht vollkommen unbrauchbar war. Daraufhin wurden die Stilanpassungen in css angepasst und eine responsive Navigationsleiste eingefügt. Das mobile Design ist nun lesbar, benötigt allerdings, insbesondere bei der Überschrift, noch weitere Feinarbeiten für ein ideales "Responsive Webdesign". Diese Veränderungen lagen leider nicht mehr im zeitlichen Rahmen.

Quellenangaben

I. Daten:

EUROSTAT:
https://ec.europa.eu/eurostat/databrowser/view/nrg_ind_rftce/default/table?lang=en&category=nrg.nrg_quant.nrg_quanta.nrg_ind_share (Thematische Daten Europa, 2021)

Opendata Rhein-Kreis-Neuss:
https://opendata.rhein-kreis-neuss.de/explore/dataset/world-administrative-boundaries-countries/export/?dataChart=eyJxdWVyaWVzIjpbeyJjb25maWciOnsiZGF0YXNldCI6Indvcmxk LWFkbWluaXN0cmF0aXZlLWJvdW5kYXJpZXMtY291bnRyaWVzIiwib3B0aW9ucyI6e319LCJjaGFydHMiOlt7ImFsaWduTW9udGgiOnRydWUsInR5cGUiOiJjb2x1bW4iLCJmdW5jIjoiQVZHIiwieUF4aXMiOiJyZWdpb25fY29kZSIsInNjaWVudGlmaWNEaXNwbGF5Ijp0cnVlLCJjb2xvciI6IiMxNjUwODAifV0sInhBeGlzIjoiaXNvM19jb2RlIiwibWF4cG9pbnRzIjo1MCwic29ydCI6IiJ9XSwidGltZXNjYWxlIjoiIiwiZGlzcGxheUxlZ2VuZCI6dHJ1ZSwiYWxpZ25Nb250aCI6dHJ1ZX0%3D&location=4,51.42661,20.25879&basemap=jawg.streets (Polygondaten der Länder in Europa)

Datensatz erneuerbare Energieproduktion Deutschland:
Manske, D., Grosch, L., & Schmiedt, J. (2022). Geo-locations and System Data of Renewable Energy Installations in Germany (Version V20210507) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.6920931

Landkreise Deutschland:
https://opendatalab.de/projects/geojson-utilities/

II. Plugins:

Leaflet-Sidebar: https://github.com/Turbo87/leaflet-sidebar

Leaflet-Pattern: https://github.com/teastman/Leaflet.pattern

Google-Piechart:
https://developers.google.com/chart/interactive/docs/gallery/piechart?hl=de#donut

Google-Table-Chart: https://developers.google.com/chart/interactive/docs/gallery/table?hl=de

Leaflet-Fullscreen: https://github.com/brunob/leaflet.fullscreen

Leaflet-Markercluster: https://github.com/Leaflet/Leaflet.markercluster

Leaflet-Control-Search: https://github.com/stefanocudini/leaflet-search

III. Icons und Schriftarten

Fontawesome: https://fontawesome.com

Google fonts: https://fonts.google.com

Mapicons: https://mapicons.mapsmarker.com