2. Funktionen

Das Internet macht erst richtig Spaß, seit die Seiten interaktiv sind. Um auf Eingaben eines Benutzers reagieren zu können, gibt es bei Programmiersprachen EventListener und Funktionen.

Konzept
Ein Ereignis („Event“) lässt sich sowohl einem Objekt zuordnen als auch als ein Objekt beschreiben. Beispielsweise ist das Ereignis „Ein Klick ist auf ein Schaltfläche erfolgt“ eine Methode, die der Schaltfläche zugeordnet werden kann. Der Klick selbst kann man als ein Objekt auffassen, das als Methode einen speziellen Programmteil (eine Funktion) ausführt. Diesen Funktionen können Werte mitgegeben werden. Diese Werte werden als Funktionsparameter oder einfach Parameter bezeichnet.
Durch EventListener können an z.B. einer Schaltfläche gleich mehrere Ereignisse bearbeitet werden. Ein Klick, ein Doppelklick, eine Mausbewegung über die Schaltfläche usw. Jedes dieser Ereignisse kann eine andere Funktion aufrufen. Insbesondere bei Karten treten mehrere solche Ereignisse auftreten, die sich auf demselben Kartenobjekt abspielen (z. B. verschieben, scrollen, Mausbewegung, Klick).

Aufgabe:

1. **Erstelle mit einem Texteditor die folgende Datei „kartenprojekt_v2-0.html“ (Änderungen zur vorherigen Datei sind rot):**

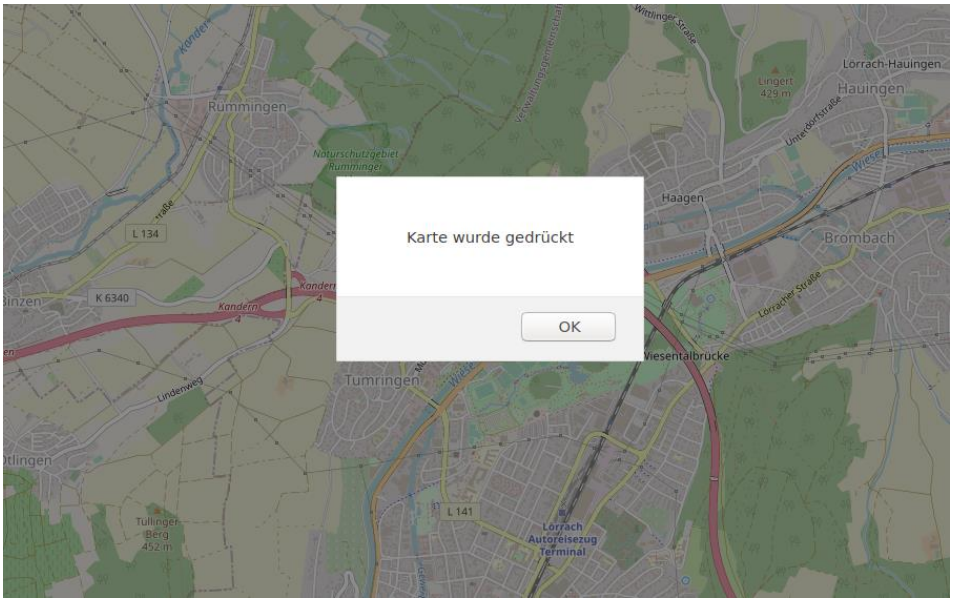
```
<!DOCTYPE html>
<html lang="de">
  <head>
     <meta charset="utf-8"/>
    <title>Mein Kartenprojekt</title>
     <script
src="https://cdn.rawgit.com/openlayers/openlayers.github.io/ma
ster/en/v5.3.0/build/ol.js"></script>
     <link rel="stylesheet"
href="https://cdn.rawgit.com/openlayers/openlayers.github.io/m
aster/en/v5.3.0/css/ol.css" type="text/css">
  </head>
  <body>
    <h2>Karte</h2>
    <div id="karte" class="karte">
</div>
     <script>
     document.getElementById("karte").addEventListener("click",
function(){alert('Kartenbereich wurde gedrückt')}, false);
     var karte = new ol.Map({
        target: 'karte',
        layers: [
          new ol.layer.Tile({
```

```
            source: new ol.source.OSM()
          }),
        ],

        view: new ol.View({
          center: ol.proj.fromLonLat([7.662277, 47.622213]),
          zoom: 14
        })
      });

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

Wird die Seite im Browser geladen und es erfolgt ein Mausklick, sollte ein Nachrichtenfenster aufgehen:



Aufgabe

2. **Im obigen Bild ist ein anderer Text zu lesen als beim Anzeigen deines Dokuments. Welche Anweisung bewirkt das Öffnen eines Mitteilungsfensters in javascript?**

Achtung, wir haben dem Objekt den EventListener angehängt. Der Abschnitt umfasst aber die gesamte Karte. Spannender wäre zu wissen, wo genau auf die Karte gedrückt wurde. Wir sollten den EventListener also besser dem Kartenobjekt selbst zuordnen.
Es gibt spezielle openlayers Eventlisteners. Für eine Klickereignis-Methode heißt die Methode einfach Objekt.on(). Der Methode on werden zwei Parameter übergeben. Der erste Parameter beschreibt das Event. Als Werte können hier „click“ oder „pointermove“ verwendet werden. Der zweite Parameter ist der Funktionsname oder die Funktion selbst.

Aufgabe

3. **Lösche den eventListener, der an das Objekt div gebunden ist und füge wie unten die Eventlistener für die Instanz „Karte“ des Objekts „Map“ ein. Ergänze die beiden Funktionen „begruessung“ und „bewegung“. Durch einen Klick auf die Karte wird im Mitteilungsfenster die Koordinaten des Mausklicks (Punkte in x-Richtung horizontal, Punkte in y-Richtung vertikal) in einem Mitteilungsfenster angezeigt. Beim Bewegen des Mauszeigers sollen die Koordinaten des Mauszeigers oberhalb der Karte angezeigt werden. Dazu musst du noch einen Abschnitt mit der id „koordinaten“ an einer geeigneten Stelle im Code anbringen. Speichere anschließend die Datei als „kartenprojekt_v2-1.html“ ab.**

```
<script>
        function begruessung(evt){
                console.log(evt);
                alert (evt.pixel);
        }
        function bewegung(evt){
                document.getElementById("koordinaten").innerHTML = evt.pixel;
        }
        var karte = new ol.Map({
      target: 'karte',
      layers: [
        new ol.layer.Tile({
          source: new ol.source.OSM()
        }),
      ],

      view: new ol.View({
        center: ol.proj.fromLonLat([7.662277, 47.622213]),
        zoom: 14
      })
    });
        karte.on('click', function(evt){begruessung(evt)});
        karte.on('pointermove', function(evt){bewegung(evt)});
        </script>
```

Aufgabe

4. **Durch Rechtsklick auf Element untersuchen und einem Klick in die Karte wird das Objekt „evt“ (event) angezeigt. Durch Klick auf den Öffnungspfeil erhältst du einen Überblick über Eigenschaften und Methoden des Objekts. Beschreibe hier den Zusammenhang der Eigenschaften coordinate und pixel.** (Tipp: coordinate gibt Informationen über die geografische Lage)

Aufgabe

5. **Die nächste Aufgabe ist schon ein kleines Projekt**

Speichere zunächst die Datei unter „kartenprojekt_v2-2.html ab. Durch Klick auf eine Schaltfläche soll die Karte auf den Punkt zentriert werden, der durch die Längen- und Breitenangabe vorgegeben ist. Wie du sicher weißt, kann jeder Punkt auf der Erdkugel durch eine Längen- und Breitenangabe identifiziert werden. Längengrade laufen durch die Pole, Breitengrade sind Parallelen zum Äquator.
Doch zurück zur Programmierung: Um einen beliebigen Längen- und Breitengrad eingeben zu können, erstellen wir an Stelle der div „koordinaten“ zwei Eingabefelder mit einem Absenden Knopf:
Der Code dazu:

```
<label for="laenge">Länge (z.B. 7.5456):</label>
<input type="text" id="laenge" name="laenge" value="7.5456">
<label for="breite">Breite (z.B. 50.234):</label>
<input type="text" id="breite" name="breite" value="50.234">
<input type="button" id=“submit“ value="Absenden">
```

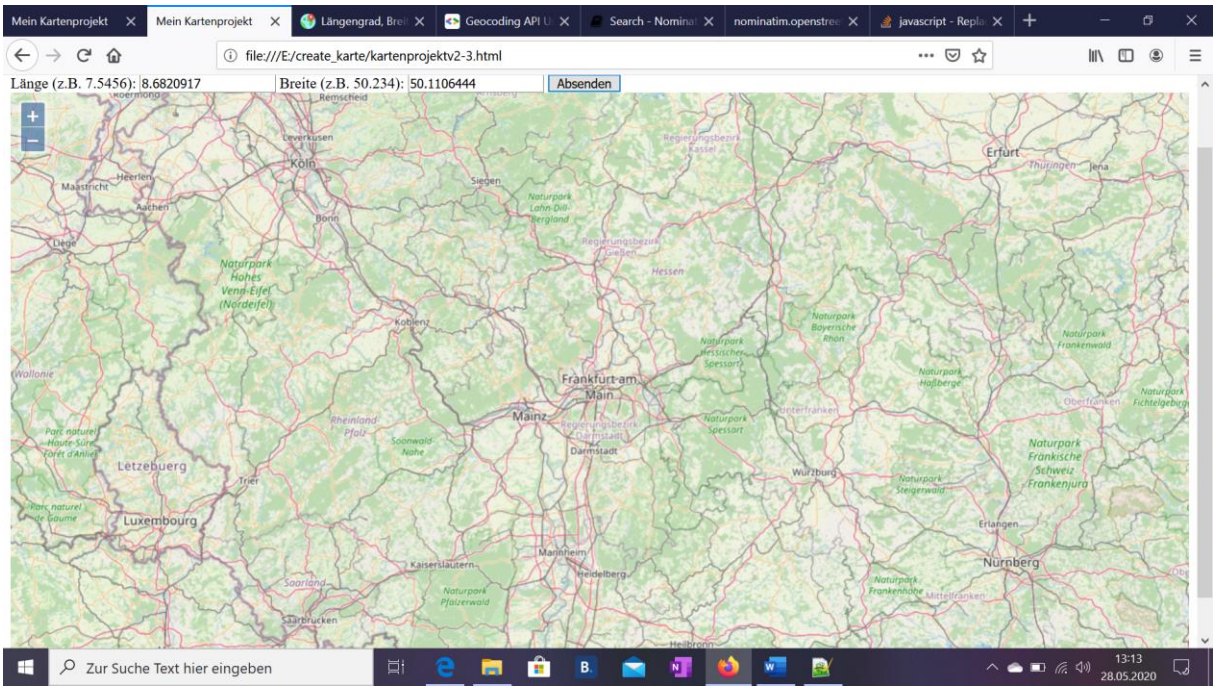
Als nächstes wird ein eventListener benötigt, der einen „Klick“ auf den Knopf „Absenden“ registriert. Den Code dazu kannst du aus Aufgabe 1 anpassen. Das Drücken des Knopfes soll die Funktion „karteZentrieren“ auslösen. Lösche eine der beiden nicht mehr benötigten Funktionen, benenne die andere in karteZentrieren um und passe sie so an:

```
var laenge = document.getElementById("laenge").value;
hier fehlt noch die entsprechende Anweisung für die Breite
karte.getView().setCenter (ol.proj.fromLonLat([parse.Float(laenge), parseFloat(breite)]));
```

Konzept:
Variablen haben einen Typ.
Stringvariablen enthalten Texte, integer-Variablen Ganzzahlen usw. Je nach dem Code (Kontext) interpretiert javascript den Variablentyp selbst. Werden zwei Variablen multipliziert, werden die Werte als Zahlen angenommen und das Ergebnis auch in einem Zahlentyp gespeichert. Man kann aber auch im Code Variablentypen ineinander umwandeln, wenn die automatisierte Umwandlung Fehler produziert.

Durch die Anweisung parse.Float(x) wird x in eine Zahlenvariable (Fließkommazahl) umgewandelt. „laenge“ und „breite“ werden zunächst als Stringvariablen gespeichert, da alle Variablen aus einem Formular (ohne weitere Angaben) als Strings gespeichert werden. Der Zahlenwert einer Stringvariablen ist immer null. ol.proj.fromLonLat erwartet eine Liste von zwei Zahlenwerten. Ohne Umwandlung übergeben wir der Funktion fromLonLat daher immer das Zahlenpaar 0,0. Um das zu vermeiden, wandeln wir das Variablenformat explizit um.

Ein Screenshot der Anwendung (mit zoom: 8):



Die Karte ist bis jetzt noch sehr langweilig, da sie bloß geografische Merkmale enthält. Thematische Karten vermitteln Informationen zu Gebieten. Hierzu fügen wir der Karte einen Layer aus dem Internet hinzu. Dazu benutzen wir einen WMS-Layer, einen hochstandardisierten Web-Map-Service.
Viele Webseiten bieten WMS-Dienste an, d.h. sie stellen thematische Karten zur Verfügung.

Aufgabe:

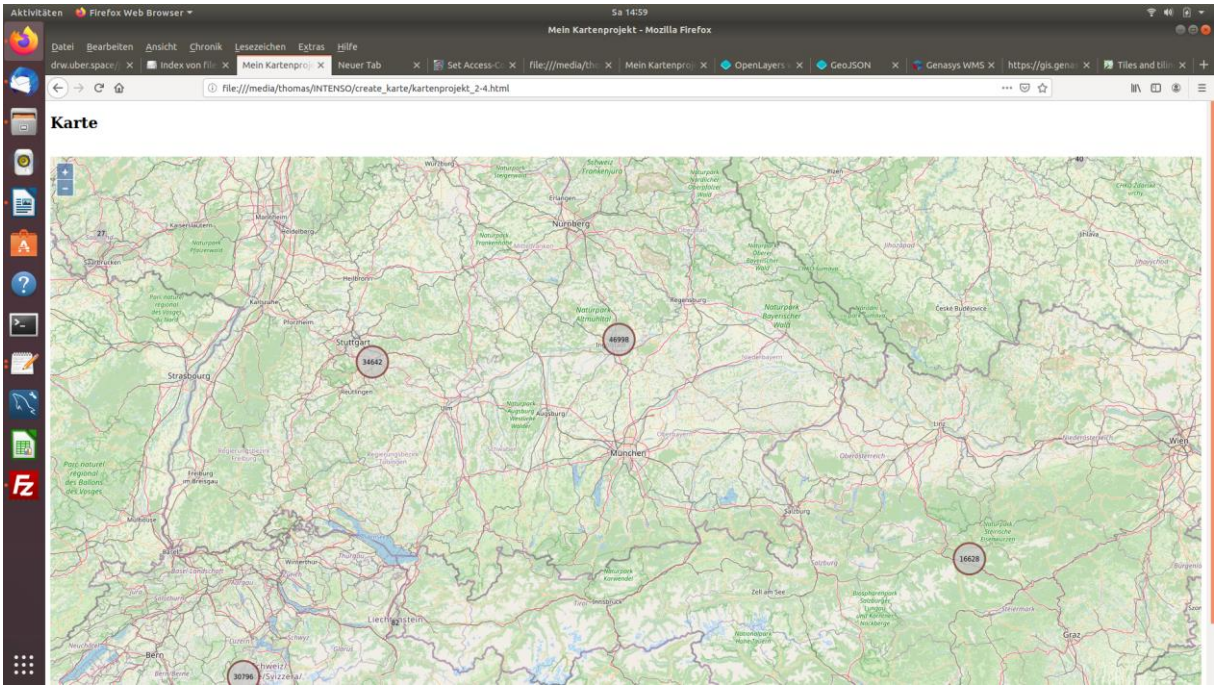
6. **Öffne „kartenprojekt_v1-2.html“ in Notepad++, gedit oder dem Editor und speichere unter dem Namen kartenprojekt_v2-3.html ab.**

Füge folgenden Code vor der Instanzerzeugung des Map-Objekts hinzu:

```
var wmsSource = new ol.source.TileWMS({
  url: 'https://gis.genasys.com/mapserver?SERVICE=WMS&REQUEST=GetMap',
  params: {'LAYERS': 'virus-confirmed',
                                'VERSION': '1.3.0',
                                'FORMAT': 'image/png',
                                'FORMAT_OPTIONS':'dpi:113',
                                'CRS':'EPSG:3857',
                                'BBOX':'-15763883.516553726,2614268.666598284,-
6230412.75033603,6527844.514799308',
                                'WIDTH':'1218',
                                'HEIGHT':'500'
          },
opacity: 0.5
});
var wmsLayer = new ol.layer.Tile({
  source: wmsSource
});
```

Jetzt muss der Layer noch der Karte zugefügt werden. Das bedeutet, dass in die Aufzählung der Layers (Innerhalb der eckigen Klammern) noch wmsLayer mit aufgezählt wird, zusätzlich zum Tile-Layer (OSM). Aufpassen, die Bezeichnung darf z.B. nicht innerhalb der source Instanzierung liegen, es müssen also erst diese Klammern geschlossen sein, es darf nur noch eine eckige Klammer offen sein. Hier ist dann die richtige Stelle (layers:[new.. ({source…}), wmsLayer]). Auch das Komma ist wichtig!

Jetzt hast du eine Karte erstellt, die die Corona Erkrankungen aktuell darstellt.



Interaktivität für die Karte:

```
var tooltip = document.createElement('div');
tooltip.setAttribute("id", "info");
var overlay = new ol.Overlay({
  element: tooltip,
  offset: [10, 0],
  positioning: 'bottom-left'
});
karte.addOverlay(overlay);

function displayTooltip(evt) {

 var viewResolution = /** @type {number} */ (karteView.getResolution());
  var url = wmsSource.getGetFeatureInfoUrl(
    evt.coordinate, viewResolution, 'EPSG:3857',
    {'INFO_FORMAT': 'text/html'});
  if (url) {
    fetch(url)
      .then(function (response) { return response.text(); })
      .then(function (html) {
          if(html) {
          overlay.setPosition(evt.coordinate);
        tooltip.innerHTML = html;
          console.log(html);
          }else {tooltip.innerHTML='';}
      });
     }
};

karte.on('pointermove', displayTooltip);
<style>
#info {
background: white;
}
</style>
```

Erklärung:

```
var tooltip = document.createElement('div');
tooltip.setAttribute("id", "info");
```

Hier wird ein Objekt (ein Abschnitt) angelegt. Dem Abschnitt wird durch die Funktion eine Eigenschaft (Attribut) zugewiesen.

```
var overlay = new ol.Overlay({
  element: tooltip,
  offset: [10, 0],
  positioning: 'bottom-left'
});
karte.addOverlay(overlay);
```

Ein Overlay-Objekt wird angelegt. Dies ist eine zusätzliche Ebene, die Hinweise zu den besonderen Objekten (Features, hier Kreise mit zahlen) enthalten kann. Du kennst das aus dem Internet, wenn z.B. Erklärungstexte über bestimmten Inhalten beim Darüberfahren angezeigt werden. Das Overlay Objekt ist normalerweise solange unsichtbar, wie keine Inhalte enthalten sind. DIe Hinweisschicht wird an der Position des Mauszeigers angelegt. Die Eigenschaften, die der neuen Instanz overlay zugewiesen werden, sind element, hier muss ein DOM-Objekt zugewiesen werden, also z.B. div. Der div-Abschnitt ist leer also wird nichts angezeigt.
Ein offset: Hier wird eine Liste erwartet. Der erste Wert gibt eine horizontale Verschiebung, der zweite eine vertikale Verschiebung des Overlay-Objekts an. Damit kann das Hinweiselement besser positioniert werden.
Abschliessend muss die Overlay-Instanz der Karte hinzugefügt werden.

```
function displayTooltip(evt) {

 var viewResolution = /** @type {number} */ (karteView.getResolution());
  var url = wmsSource.getGetFeatureInfoUrl(
    evt.coordinate, viewResolution, 'EPSG:3857',
    {'INFO_FORMAT': 'text/html'});
  if (url) {
    fetch(url)
      .then(function (response) { return response.text(); })
      .then(function (html) {
          if(html) {
          overlay.setPosition(evt.coordinate);
        tooltip.innerHTML = html;
          console.log(html);
          }else {tooltip.innerHTML='';}
      });
     }
};

karte.on('click', displayTooltip);
```

Durch den Eventlistener wird durch Klick auf die Karte die Funktion displayTooltip, die man auch gern zeigeTooltip nennen kann.
Um die Informationen zu den Bildelementen zu erhalten, muss der Server kontaktiert werden. Da die Bildelemente in unterschiedlichen Vergrößerungsstufen an anderen Stellen in den Bildern auftauchen, muss noch die Auflösung angegeben werden. Wenn ein Pixel 5 km entspricht, hab ich die Stelle des Bildelements in diesem Umkreis (von sagen wir 10 Pixeln also 50 km) getroffen, bei 1 km pro Pixel, muss mein Klick innerhalb von 10 km erfolgen.
Wenn es eine url gibt (if(url), dazu muss die WMS-Quelle kontaktiert werden), wird das Ergebnis geholt (fetch(url)). Die folgende Syntax ist kompliziert, es handelt sich um eine promise.
Im Ergebnis bewirkt der code, dass falls eine Antwort vom Server erhalten wird (also ein Feature angeklickt wurde), das Ergbnis in der Variablen html gespeichert wird. In der Regel werden bei WMS-Layern hier kleine Tabellen produziert.
Diese Tabellenbeschreibung wird jetzt in den Tooltip-Abschnitt geschrieben:
tooltip.innerHTML = html;

Über overlay.setPosition(evt.coordinate); wird die Position des Overlays-Objekts festgelegt.
Liefert der Server nichts zurück (else) wird durch Zuweisung eines leeren Inhalts in das tooltip-Element ein bisheriger Inhalt gelöscht. Dadurch kann man ein Tooltip durch Klicken auf eine leere Stelle in dem WMS-Layer unsichtbar machen. Ansonsten bleibt der Tooltip so lange sichtbar, bis ein anderes Element angeklickt wird.
Die Style-Anweisung gehört in den <head></head> Abschnitt der HTML-Seite (zwischen <style>-Start- und End-Tags) und gibt an, dass der Hintergrund weiß sein soll.

Aufgabe

7. **Probiere andere WMS einzubinden, die im Internet zu finden sind, z.B. Regionalatlas des Statistischen Bundesamts mit Informationen zu Umwelt, Einkommen, Wahlen und vielem anderem.** Hier kann man auch Legenden einbinden. getGetFeatureInfo ist hier allerdings blockiert (Anti-CORS-Standard).