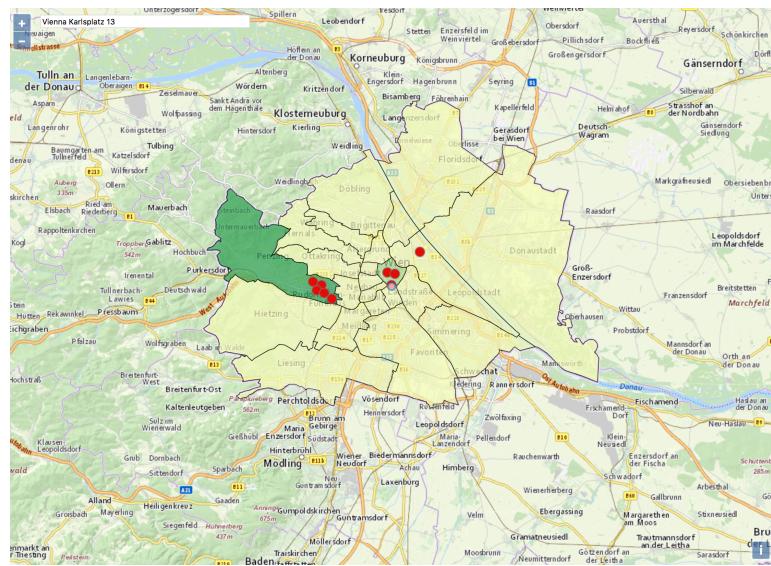


geoweb.m10 – Hands-on Client-Server Applikation



1 Client, Server, Service?.....	2
1.1 Wann ist ein eigener Serverbetrieb sinnvoll?.....	2
1.2 Arten von Services.....	2
2 Feedback-Karte	2
2.1 Kurzbeschreibung der Applikation.....	2
2.2 Anforderungen.....	3
3 Vorbereitung der Datenbasis	3
3.1 Geometriespalte für die Feedback-Tabelle	3
3.2 Wahl einer geeigneten Methode für die Statistik-Funktion.....	4
4 Eingeben eines neuen Feedbacks durch Klick in die Karte	4
4.1 Position zum Feedback-Formular hinzufügen.....	4
4.2 Übergebene Position in der Datenbank speichern	5
4.3 Bei Klick in die Karte Feedback-Formular aufrufen	5
4.4 Bei Rückkehr zur Karte wieder die vorherige Position anzeigen.....	5
5 Bereits abgegebenes Feedback in der Karte anzeigen	6
5.1 Daten als GeoJSON bereitstellen.....	6
5.2 Feedback-Layer zu OpenLayers Karte hinzufügen	7
5.3 Popup bei Klick auf Feedback-Position	7
6 Statistik – Bezirke nach Anzahl gegebener Feedbacks.....	9
6.1 Berechnung der Feedbacks pro Bezirk	9
7 Deployment	11
8 Referenz	11

1 Client, Server, Service?

Die Geschichte der IT erlebte bereits mehrere Paradigmenwechsel bezüglich der Frage, was am Client (heute meist Browser) und was am Server laufen soll. Mit der steigenden Popularität von Web-Services kommt noch die Frage dazu, ob man eigene Server betreiben oder auf vorhandene Services zugreifen soll. Lösungen, die eigene Client-Software, aber keine eigene Server-Software beinhalten, werden in diesem Kontext als „serverless“ bezeichnet. Die Karte aus Aufgabe A4 wäre eine solche „serverless“ Applikation: eine rein clientseitige Kartenanwendung, ergänzt um eine Ortssuche, welche Informationen von einem Service (photon) bezieht.

1.1 Wann ist ein eigener Serverbetrieb sinnvoll?

Das Betreiben von Servern bringt immer einen Administrationsaufwand mit sich. Es ist daher abzuwägen, ob die Kosten der Administration höher oder niedriger sind als die eines geeigneten Services. Ein weiteres Kriterium ist Datenhoheit: können/dürfen meine Daten bei Drittanbietern gespeichert werden?

1.2 Arten von Services

Im Zuge dieser Lehrveranstaltung haben wir ausschließlich Web-Services kennengelernt, die direkt vom Client aus über HTTP angesprochen werden. Es gibt jedoch zunehmend Services, die einem eigentlichen Serverbetrieb ähnlich sind. Meist meint man damit Cloud Services, wie z.B. die Amazon Web Services (AWS, <https://aws.amazon.com/>).

Solche Cloud-Services werden meist stundenweise nach Nutzungszeit bezahlt. Man unterscheidet meist zwischen Services die Rechenzeit, Speicherplatz oder Datenbanken bieten.

Die Services am IFIP-Studentenserver, die im Rahmen dieser Lehrveranstaltung angeboten werden, könnte man als Speicherplatz- und Datenbank-Services bezeichnen: Speicherplatz für Webseiten (einschließlich dynamische, mit PHP) und eine PostgreSQL+PostGIS Datenbank.

Wenn wir hier von Client-Server Anwendungen sprechen, meinen wir also Anwendungen, die zwar Serverkomponenten, aber keinen eigenen Serverbetrieb beinhalten.

2 Feedback-Karte

2.1 Kurzbeschreibung der Applikation

Die Applikation, die wir erstellen wollen, baut auf dem Feedback-Formular aus Block 2 und der Karte aus Block 4 auf. Statt jedoch das Feedback auf einer eigenen Seite in einem Formular einzugeben, wollen wir das Feedback in einer Karte eintragen, in der das Feedback mit einem Ort verknüpft wird. Bereits abgegebene Feedbacks sollen in der Karte interaktiv angezeigt werden, und es soll eine statistische Auswertung der abgegebenen Feedbacks nach Bezirk geben.

2.2 Anforderungen

Konkrete Anforderungen der Applikation sind:

- Eingeben eines neuen Feedbacks durch Klick in die Karte.
- Interaktives Anzeigen bestehenden Feedbacks in der Karte: Darstellung der Feedback-Punkte mit Detailinformation bei Klick auf einen Punkt.
- Statistik-Funktion mit einer einfachen Auswertung: Anzahl der Feedback-Punkte nach Wiener Gemeindebezirk

3 Vorbereitung der Datenbasis

Erster Schritt für die Umsetzung der Applikation ist die Vorbereitung der Datenbasis:

- Hinzufügen einer Geometriespalte zur Feedback-Tabelle
- Wahl einer geeigneten Methode für die Statistik-Funktion

3.1 Geometriespalte für die Feedback-Tabelle

Die bestehende Feedback-Tabelle (`gxx_feedback`) muss um eine Geometrie-Spalte erweitert werden. In dieser Spalte soll die dem Feedback zugeordnete Position gespeichert werden.

PostGIS stellt für Geometrien den Datentyp `geometry` zur Verfügung. Als Name für Geometrie-Spalten wird häufig `the_geom` verwendet. Mit phppgadmin können wir leicht eine solche Spalte anlegen:

- Tabelle `gxx_feedback` im Baum anklicken
- „Add column“ anklicken
- Spalte mit Namen `the_geom` und Typ `geometry` anlegen



Nach dem Anlegen können wir sofort testen, ob alles funktioniert hat, indem wir händisch zu einem bestehenden Feedback-Eintrag eine Position hinzufügen:

- Tabelle `gxx_feedback` im Baum anklicken
- „Browse“ anklicken
- In einer beliebigen Zeile „Edit“ klicken
- In der `the_geom` Zeile bei Format „Expression“ auswählen und bei Value `ST_GeomFromText('POINT(16 48)')` eingeben.
- Mit „Save“ speichern.

f_datum	text	Value	<input type="text" value="24-10-2017"/>
the_geom	geometry	Expression	<input type="text" value="ST_GeomFromText('POINT(16.37 48.20)')"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input checked="" type="checkbox"/> Enable AutoComplete			

Wenn das funktioniert, ist die Datenbank fertig eingerichtet für die neue Aufgabe.

3.2 Wahl einer geeigneten Methode für die Statistik-Funktion

Die Berechnung der eingegebenen Feedbacks pro Bezirk kann entweder in der Datenbank oder im Browser erfolgen. Was spricht für welche Methode? Ein Anhaltspunkt ist folgender: wenn die Daten ohnehin bereits am Client vorliegen, empfiehlt sich eine Berechnung am Client. Denn dann werden keine unnötigen Ressourcen am Server beansprucht. Genau das wird bei uns der Fall sein: sowohl die Bezirksgrenzen als auch die Feedback-Punkte werden für die Kartendarstellung benötigt, sind also bereits im Browser verfügbar. Die Berechnung wird also in JavaScript durchgeführt werden, sobald Bezirke und Feedbacks geladen sind.

Wäre das Szenario ein anderes (also z.B. keine Darstellung der Bezirksgrenzen, dafür Bezirksgrenzen in der Datenbank gespeichert, oder eine Darstellung, wo nicht alle Feedback-Punkte in der Karte ersichtlich sind), könnte eine Berechnung am Server sinnvoller sein. Die Methode wäre dann sinnvollerweise eine Spatial SQL Abfrage (Spatial Join), unter Verwendung der PostGIS-Funktion ST_Contains. Beispiel:

```
SELECT bezirke.name, count(*) AS anzahl FROM bezirke, gxx_feedback WHERE
ST_Contains(bezirke.geom, gxx_feedback.geom) GROUP BY bezirke.id;
```

4 Eingeben eines neuen Feedbacks durch Klick in die Karte

Aufbauend auf dem bestehenden Feedback-Formular wollen wir ein Feedback mit einer Position verknüpfen. Der einfachste Lösungsansatz dafür ist die Verwendung der bestehenden Formularseite, die dann jedoch per JavaScript aufgerufen wird. An die URL wird die geklickte Position als sogenannter Query-Parameter angehängt. Also z.B.

<https://student.ifip.tuwien.ac.at/geoweb/2017/gxx/map/feedback.php?pos=16.37%2048.20>. Daraus kann in PHP ein Formularfeld generiert werden, welches dann einfach mit abgeschickt wird.

Damit das Klicken des Zurück-Buttons zur Karte mit der zuletzt gezeigten Position führt, werden wir weiters ein kleines OpenLayers-Tool einsetzen: <https://www.npmjs.com/package/ol-hashed>.

4.1 Position zum Feedback-Formular hinzufügen

Derzeit ist das Feedback-Formular eine reine html-Seite (siehe Beispiel 6 der Php-Kurzeinführung auf <https://student.ifip.tuwien.ac.at/geoweb/intranet/intern/index.htm>). Wenn wir daraus eine php-Seite machen, können wir die übergebene Position (`pos`) direkt ins Formular übernehmen. Dazu fügen wir einfach als erste Zeile innerhalb des `<form>` Elements eine Zeile mit einem versteckten Formularelement hinzu:

```
<input type="hidden" name="pos" value="<?php echo $_GET['pos'];?>">
```

Dann entfernen wir noch den Hinweis, dass das Feedback per e-mail an die Autoren/innen zugestellt wird, und speichern die Datei als `feedback.php` ab (nicht im Intranet, sondern in einem frei zugänglichen Verzeichnis).

4.2 Übergebene Position in der Datenbank speichern

Nun ändern wir die Datei `feedback_send.php`, und speichern die neue geänderte Version (ohne e-mail Funktion und ohne Hinweis daruf!) im gleichen Verzeichnis wie `feedback.php` ab. Wir müssen das Formularfeld `pos` auswerten, und das Ergebnis ins `the_geom` Feld der Datenbank speichern. Dafür verwenden wir die PostGIS Funktion `ST_GeomFromText`. Wir ändern einfach die Zeilen, welche den SQL-Befehl erzeugen:

```
$sql = "INSERT INTO gxx_feedback (f_name,f_email,f_anrede,f_msg,f_geoweb,f_datum,the_geom)";
$sql = $sql . " VALUES ('" . $name . "','" . $email . "','" . $anrede .
"','" . $message . "','" . $teamflag . "','" . date("d-m-Y") .
"',ST_GeomFromText('POINT(" . $_POST['pos'] . ")'))";
```

4.3 Bei Klick in die Karte Feedback-Formular aufrufen

Dazu ändern wir die JavaScript-Applikation aus Aufgabe A4. In der Datei `main.js` fügen wir einen sogenannten Listener für Klick-Events in der Karte hinzu. Die folgenden Zeilen werden dazu einfach am Ende der bestehenden Datei angefügt:

```
map.on('singleclick', function(e) {
  var pos = proj.toLonLat(e.coordinate);
  window.location.href =
    'https://student.ifip.tuwien.ac.at/geoweb/2017/gxx/map/feedback.php?pos=' +
    pos.join(' ');
});
```

Falls das dafür benötigte Modul `proj` noch nicht importiert ist, auch am Anfang der Datei noch den benötigten Import hinzufügen:

```
import proj from 'ol/proj';
```

Zweck des hinzugefügten Codes ist es, bei Klick in die Karte die Feedback-Seite aufzurufen, und zwar mit der geklickten Position in der URL.

4.4 Bei Rückkehr zur Karte wieder die vorherige Position anzeigen

Standardmäßig reagiert OpenLayers nicht auf den Zurück-Button des Browsers. Mit dem Utility `ol-hashed` können wir das ändern. Mit `npm install ol-hashed` wird das Paket unserer `package.json` Datei hinzugefügt. Dann wird das Modul in die `main.js` Datei importiert mit

```
import sync from 'ol-hashed';
```

Mit einer einzigen Zeile JavaScript Code wird die Funktionalität aktiviert. Diese Zeile wird einfach am Ende der Datei angefügt:

```
sync(map);
```

Der Effekt ist, dass nun in der URL der Kartenseite der aktuelle Kartenmittelpunkt und die Zoomstufe mitgespeichert wird, und es wird ein Rückkehrpunkt für den Zurück-Button des Browsers gesetzt.

Nach Erledigung dieser Schritte können wir in der Kartenapplikation bereits Feedback eingeben, das mit einer Position in der Karte verknüpft ist. Im nächsten Kapitel werden wir die Orte von bereits abgegebenem Feedback in der Karte darstellen und den Feedback-Text interaktiv bereitstellen.

5 Bereits abgegebenes Feedback in der Karte anzeigen

5.1 Daten als GeoJSON bereitstellen

Wie wir bereits wissen, kann OpenLayers sehr einfach Daten im GeoJSON Format darstellen. Wir werden also PHP verwenden, um die bereits abgegebenen Feedback-Einträge aus der Datenbank zu holen und als GeoJSON anzuzeigen.

Den folgenden PHP-Code speichern wir als `postgis_geojson.php` im gleichen Verzeichnis wie zuvor die `feedback.php` Datei ab:

```
<?php
include 'geoweb_pg_open.php';
# Build SQL SELECT statement and return the geometry as a GeoJSON element
$sql = 'SELECT *, public.ST_AsGeoJSON(the_geom, 6) AS geojson FROM gxx.gxx_feedback';
# Try query or error
$result = pg_query($sql);
if (!$result) {
    echo "An SQL error occurred.\n";
    exit;
}
# Build GeoJSON feature collection array
$geojson = array(
    'type' => 'FeatureCollection',
    'features' => array()
);
# Loop through rows to build feature arrays
while ($row = pg_fetch_assoc($result)) {
    $properties = $row;
    # Remove geojson and geometry fields from properties
    unset($properties['geojson']);
    unset($properties['the_geom']);
    $feature = array(
        'type' => 'Feature',
        'geometry' => json_decode($row['geojson'], true),
        'properties' => $properties
    );
    # Add feature arrays to feature collection array
    array_push($geojson['features'], $feature);
}
header('Access-Control-Allow-Origin: *');
header('Content-type: application/json; charset=utf-8');
echo json_encode($geojson, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES |
    JSON_NUMERIC_CHECK);
include 'geoweb_pg_close.php';
?>
```

Der Code dieses Skripts kommt in abgewandelter Form von <https://github.com/bmcbride/PHP-Database-GeoJSON>. Im Prinzip macht dieses PHP-Skript dasselbe wie Beispiel 8 aus der Php-PostgreSQL Einführung (<https://student.ifip.tuwien.ac.at/geoweb/intranet/intern/index.htm>). Der

Unterschied ist die Datenstruktur, welche den GeoJSON-Konventionen entspricht (d.h. mit „FeatureCollection“ und „Feature“).

Zusätzlich zum Content-type Header senden wir auch noch einen Access-Control-Allow-Origin Header. Dieser erlaubt JavaScript-Applikationen den Zugriff auf die Daten.

5.2 Feedback-Layer zu OpenLayers Karte hinzufügen

In Block 4 haben wir Maputnik (<https://maputnik.github.io/>) verwendet, um die Styles für unsere Karte zu konfigurieren. Auf die daraus erzeugte style.json Datei haben wir bisher über eine GitHub URL (<https://gist.githubusercontent.com/anonymous/efb8d2ab014796b45817954ed4e188ba/raw/4e15466382a261a6e7da9cefc34c5dc05f28525/style.json>) zugegriffen. Nun wollen wir diese Datei um einen Layer, nämlich die Feedback-Positionen, erweitern. Zunächst speichern wir die Datei im Ordner data unserer Kartenapplikation ab und ändern den Pfad in main.js auf "data/style.json".

Nun fügen wir einen Eintrag unter "sources" hinzu:

```
"feedback": {  
    "type": "geojson",  
    "data": "https://student.ifip.tuwien.ac.at/geoweb/2017/gxx/map/postgis_geojson.php"  
},
```

Schließlich brauchen wir noch einen zusätzlichen Eintrag unter "layers":

```
{  
    "id": "feedback",  
    "type": "circle",  
    "source": "feedback",  
    "paint": {  
        "circle-radius": 7,  
        "circle-color": "rgba(232, 12, 12, 1)",  
        "circle-stroke-color": "rgba(127, 127, 127, 1)",  
        "circle-stroke-width": 1  
    }  
}
```

Alternativ können diese Einträge auch mit der grafischen Oberfläche von Maputnik erstellt werden.

5.3 Popup bei Klick auf Feedback-Position

Bisher reagieren wir bei Klick in die Karte immer mit dem Öffnen des Feedback-Formulars. Nun wollen wir die Applikation ändern, sodass bei Klick auf eine Feedback-Position stattdessen ein Info-Popup mit dem Feedback angezeigt wird. Für die zusätzliche Funktionalität verwenden wir im Wesentlichen Code aus dem OpenLayers-Workshop (<http://openlayers.org/workshop/en/vectortile/interact.html>). Das zusätzliche CSS und HTML für unsere index.html Datei übernehmen wir 1:1 von dort.

CSS (einfügen im style Block):

```

.arrow_box {
  position: relative;
  background: #000;
  border: 1px solid #003c88;
}
.arrow_box:after, .arrow_box:before {
  top: 100%;
  left: 50%;
  border: solid transparent;
  content: " ";
  height: 0;
  width: 0;
  position: absolute;
  pointer-events: none;
}
.arrow_box:after {
  border-color: rgba(0, 0, 0, 0);
  border-top-color: #000;
  border-width: 10px;
  margin-left: -10px;
}
.arrow_box:before {
  border-color: rgba(0, 60, 136, 0);
  border-top-color: #003c88;
  border-width: 11px;
  margin-left: -11px;
}
.arrow_box {
  border-radius: 5px;
  padding: 10px;
  opacity: 0.8;
  background-color: black;
  color: white;
}
#popup-content {
  max-height: 200px;
  overflow: scroll;
}
#popup-content th {
  text-align: left;
  width: 125px;
}

```

HTML (einfügen nach dem map-container div):

```

<div class="arrow_box" id="popup-container">
  <div id="popup-content"></div>
</div>

```

Im JavaScript Code in main.js fügen wir den Code für das Overlay (d.h. das Popup) hinzu, der ebenfalls unverändert aus dem Workshop-Beispiel kommt:

```

var overlay = new Overlay({
  element: document.getElementById('popup-container'),
  positioning: 'bottom-center',
  offset: [0, -10],
  autoPan: true
});
map.addOverlay(overlay);
overlay.getElement().addEventListener('click', function() {
  overlay.setPosition();
});

```

Der Code kann einfach am Ende der Datei hinzugefügt werden. Und damit er funktioniert, muss auch das Overlay Modul aus dem ol Paket importiert werden (am Anfang der Datei main.js):

```
import Overlay from 'ol/overlay';
```

Schließlich müssen wir noch den bestehenden `singleclick` Listener in `main.js` abändern, sodass zuerst auf Feedback-Positionen geprüft wird, bevor das Feedback-Formular aufgerufen wird. Wenn es an der geklickten Position ein Feedback gibt, wird das Popup geöffnet. Andernfalls das Feedback-Formular. Der komplette neue Listener sieht dann so aus:

```
map.on('singleclick', function(e) {
    var markup = '';
    map.forEachFeatureAtPixel(e.pixel, function(feature) {
        var properties = feature.getProperties();
        if (properties['f_nr']) {
            markup += `${markup} && '<hr>'`}<table>;
            for (var property in properties) {
                if (property != 'geometry') {
                    markup += `<tr><th>${property}</th><td>${properties[property]}</td></tr>`;
                }
            }
            markup += `</table>`;
        });
        if (markup) {
            document.getElementById('popup-content').innerHTML = markup;
            overlay.setPosition(e.coordinate);
        } else {
            overlay.setPosition();
            var pos = proj.toLonLat(e.coordinate);
            window.location.href =
                'https://student.ifip.tuwien.ac.at/geoweb/2017/gxx/map/feedback.php?pos=' +
                pos.join(' ');
        }
    });
});
```

6 Statistik – Bezirke nach Anzahl gegebener Feedbacks

Im letzten Arbeitsschritt wollen wir nun die Daten der abgegebenen Feedbacks und der Bezirkspolygone so verknüpfen, dass wir die Bezirke nach der Anzahl der abgegebenen Feedbacks einfärben können.

6.1 Berechnung der Feedbacks pro Bezirk

Voraussetzung für diese Berechnung ist, dass die Daten fertig geladen sind. Wir erinnern uns: das Laden von Daten im Webbrowser erfolgt asynchron. Vektor-Layer in OpenLayers sind fertig geladen, wenn deren `source` ihren `change` Listener zum ersten Mal aufruft. Da die Layer über die `style.json` Datei definiert werden, müssen wir sie suchen, indem wir auf die `layers` Collection der `map` einen `add` Listener ansetzen.

Sobald wir beide Layer haben und ihre Daten geladen sind, können wir zur Berechnung schreiten. Wir ermitteln einfach für jeden Feedback-Punkt den Bezirk, in dem er liegt, und erhöhen den Wert im von uns neu erzeugten ‚FEEDBACKS‘ Attribut des jeweiligen Bezirks:

```
var loadedSources = {};
map.getLayers().on('add', function(e) {
    var source = e.element.getSource();
    if (source instanceof VectorSource) {
        source.once('change', function() {
            loadedSources[source.getUrl()] = source;
            if (Object.keys(loadedSources).length == 2) {
                var feedbacks =
                    loadedSources['https://student.ifip.tuwien.ac.at/geoweb/2017/gxx/map/postgis_geojson.php'].getFeatures();
```

Nun müssen wir nur noch den Style für die bezirksgrenzenogd Source ändern. Zum Beispiel so, dass der Bezirk umso dunkler dargestellt wird, je mehr Feedbacks abgegeben wurden. Dies kann wieder in Maputnik geschehen, oder manuell durch Editieren der Datei style.json im data Verzeichnis. Der geänderte Style für die Layer, welchen die Source bezirksgrenzenogd verwenden, kann zum Beispiel so aussehen:

```
{
  "id": "low-feedback",
  "type": "fill",
  "source": "bezirksgrenzenogd",
  "filter": [
    "all",
    [
      "<=",
      "FEEDBACKS",
      1
    ]
  ],
  "paint": {
    "fill-color": "rgba(247, 252, 185, 1)",
    "fill-outline-color": "rgba(4, 4, 4, 1)",
    "fill-opacity": 0.7
  }
},
{
  "id": "mid-feedback",
  "type": "fill",
  "source": "bezirksgrenzenogd",
  "filter": [
    "all",
    [
      ">",
      "FEEDBACKS",
      1
    ],
    [
      "<",
      "FEEDBACKS",
      5
    ]
  ],
  "paint": {
    "fill-color": "rgba(173, 221, 142, 1)",
    "fill-outline-color": "rgba(4, 4, 4, 1)",
    "fill-opacity": 0.7
  }
},
{
  "id": "high-feedback",
  "type": "fill",
```

```

    "source": "bezirksgrenzenogd",
    "minzoom": 0,
    "filter": [
      "all",
      [
        ">=",
        "FEEDBACKS",
        5
      ]
    ],
    "paint": {
      "fill-color": "rgba(49, 163, 84, 1)",
      "fill-outline-color": "rgba(4, 4, 4, 1)",
      "fill-opacity": 0.7
    }
  },

```

Für eine ansprechende Farbskala der 3 Klassen habe ich <http://colorbrewer.org/> verwendet.

7 Deployment

An dieser Stelle sollten alle PHP-Skripts bereits am Server liegen. Der JavaScript-Client muss nun auch noch auf den Server kopiert werden. Dazu erstellen wir zunächst ein Build:

```
npm run build
```

Dieser Schritt erzeugt eine Verzeichnisstruktur im Verzeichnis `build`. Die komplette Verzeichnisstruktur sollte nun per FTP ins selbe Server-Verzeichnis wie die php Dateien kopiert werden. Da die HTML-Datei `index.html` heißt, wird die Applikation automatisch beim Annavigieren des Server-Verzeichnisses geöffnet.

8 Referenz

Sämtliche im Zuge dieser Hands-on Übung erzeugten Dateien sowie eine Kurzanleitung zur Installation sind auf <https://github.com/ahocevar/tuwien-geoweb-2017/> zu finden.