Admin-Dokumentation zur Kartenanwendung Ikrosmap

Erstellt von: Dr. Peter Korduan, GDI-Service
letzte Änderung am: 31.05.2017

Änderungen:

Datum	Änderung
31.05.2017	Aufbau, Installation und Konfiguration hinzugefügt

Inhaltsverzeichnis

1 Aufbau der Anwendung

Die Anwendung basiert auf folgenden Komponenten, siehe Tabelle 1.

Komponente	Beschreibung
WFS	Dienste, die Daten liefern, die in der Anwendung dargestellt werden sollen
wfs2json	Web-Anwendung, die ein GML-Dokument von einem WFS in eine JSON-Datei umwandelt.
json2index	Web-Anwendung, die aus einer JSON-Datei einen Suchindex erzeugt.
web-App	Die Web-Site, die auf dem Web-Server bereitgestellt wird
Externe Bibliotheken	JavaScript Bibliotheken, die in der Web-Site verwendet werden, aber nicht im Repository Ikrosmap verwaltet werden.
web-Client	Browser, die die Web-Anwendung anzeigen

Tabelle 1: Beschreibung der Komponenten

Ein oder mehrere WFS liefert die Daten, die in der Anwendung zu sehen sind. Die WFS müssen aber für den Betrieb der Anwendung nicht zwangsweise ständig laufen, weil der Workflow so gewählt wurde, dass aus den WFS zunächst JSON-Dateien (JSON-Datei und INDEX-Datei) erzeugt werden. Diese werden dann in die Web-Anwendung geladen. Hat sich der Inhalt eines WFS geändert, oder soll eine neue Version der Daten angezeigt werden, müssen nur die JSON-Dateien überschrieben und die Web-Anwendung im Client neu geladen werden.

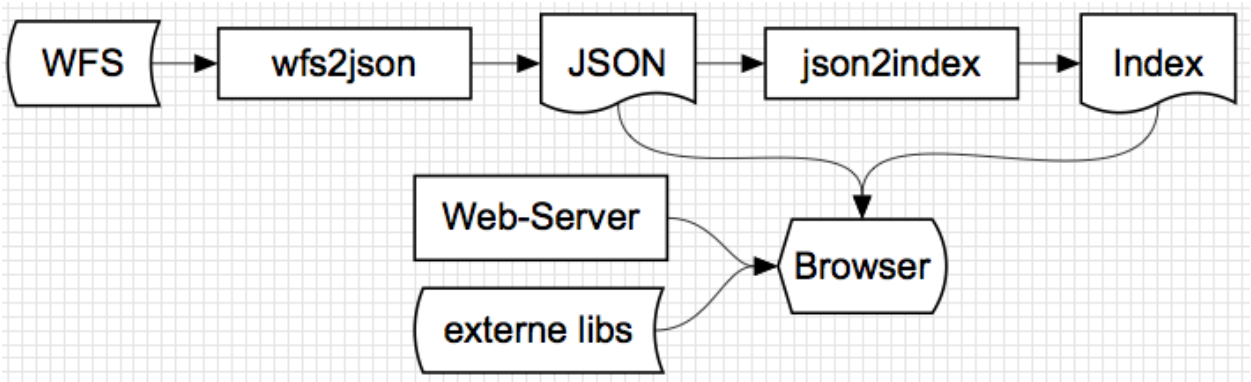

Abbildung 1: Zusammenhang der Komponenten

Die Web-Anwendung wird über einen Apache zur Verfügung gestellt. Der Browser läd externe Bibliotheken und holt sich über AJAX-Requests die benötigten JSON-Daten und Index-Daten. Die Anwendung hat die in Abbildung 2 dargestellte Verzeichnisstruktur. Die Beschreibung der Verzeichnisse und Dateien findet sich in Tabelle 2.

Verzeichnis/Datei	Inhalt
index.html	Startdatei. Kann als Home für die Kartenanwendung verwendet werden. Die einzelnen Seiten in app funktionieren auch ohne die index.html
apps	HTML-Seiten, die die Anwendung konfigurieren.
css	CSS Style-Dateien, Pro controller eine Datei
doc	Dokumentationen für Administration und Nutzer
img	Bilder und Icons für die Marker und Legende
js	JavaScript-Dateien
js/app.js	Datei zum Laden der Kopfdateien und Initialisieren der Anwendung
js/controller	Laden der Daten und Definition der Funktionalität und der Eventhandler. Je ein Controller für die Kartenanzeige, das Geocoding, das Routing und für die Hilfe.
Js/controls	Definitionen von Controls, die in die Karte eingebunden werden.
js/models	Definition der in der Karte angezeigten Features und der Controls.
js/views	Pro Controller gibt es verschiedene views zur Darstellung im Browser. Die Views beinhalten eine Zuweisung zur Variable html mit HTML-Text. Der Text kann auch Variablen beinhalten.
LICENSE	Beinhaltet Information über die Softwarelizenz
README.md	Enthält Metadaten zum git Repository. Der Text wird auf Github zur Beschreibung des Softwareprojektes angezeigt.
osm2poserviceproxy.php	PHP-Datei zur Umleitung von Anfragen an den lokalen Server auf einen Routing-Dienst osm2po

Tabelle 2: Beschreibung der Verzeichnis/Datei-Struktur

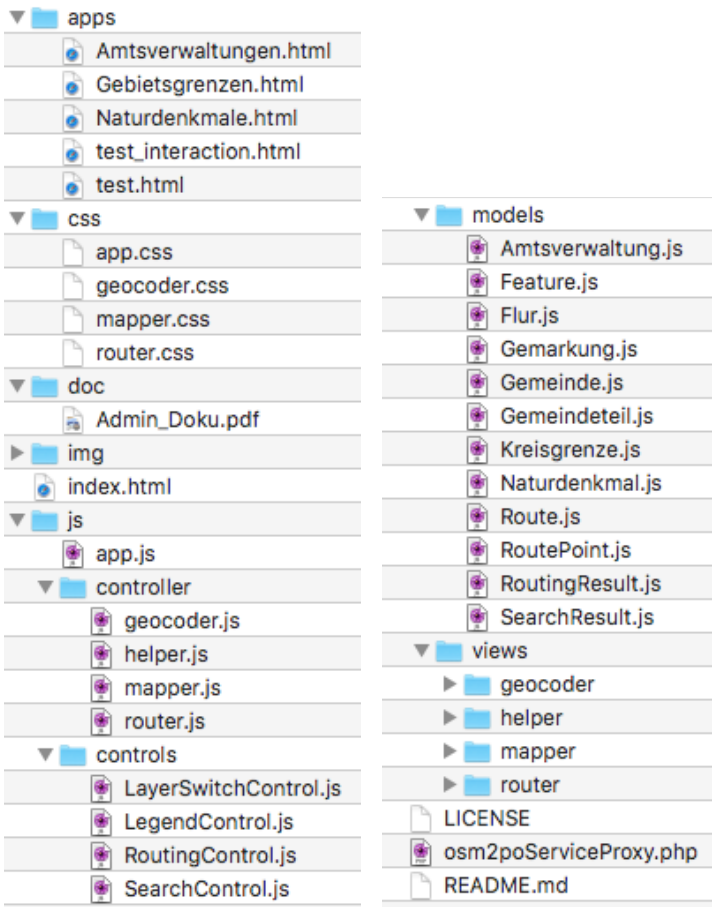

Abbildung 2: Verzeichnisstruktur der Anwendung

Das Laden der Anwendung erfolgt in folgender Reihenfolge:

- Aufruf einer Anwendung aus dem Ordner apps, z.B. Naturdenkmale.html
- Lesen der Konfiguration LkRosMap.config
- Einbinden der JavaScript-Datei js/app.js
- Ausführen der Funktion LkRosMap.init, die in js/app.js definiert ist
  - Einstellung der Projektion
  - Initialisierung der controller jeweils durch LkRosMap.controller.<name>.init()

Beim Initialisieren im mapper controller wird folgendes ausgeführt:

- Laden der views, die im controller verwendet werden
- Erzeugen der TileLayer
- Erzeugen der VectorLayer
- Initialisieren des OpenLayers Map Objektes
- Beschränkung auf extent der Layer
- Initialisieren des Info-Fensters
- Setzen der Eventhandler

2 Installation

Benötigt wird ein Apache Web Server und Browser zum Betrachten der Anwendung.

2.1 Apache

Um die Anwendung im Web verfügbar zu machen muss das Repository lkrosmap angelegt werden und ein

Alias in der Apache-Konfiguration eingefügt, der auf das Verzeichnis lkrosmap verweist. Damit ist die Anwendung über http://meinserver.de/alias verfügbar. Beispiel:

Anlegen des Repositories:

```
cd /var/docker/www/apps
git clone https://github.com/pkorduan/lkrosmap.git
```

Anlegen einer Konfigurationsdatei für Apache

```
vi /home/gisadmin/etc/apache2/sites-available/lkrosmap.conf
```

In die Datei kommt der folgende Text

```
Alias /lkrosmap "/var/www/apps/lkrosmap/"
<Directory "/var/www/apps/lkrosmap/">
  AllowOverride None
  Options FollowSymLinks Multiviews
  Order allow,deny
  Allow from all
</Directory>
```

Verfügbarmachen der config-Datei

```
cd  /home/gisadmin/etc/apache2/sites-enabled
ln -s ../sites-available/lkrosmap
```

Neuladen der Apache-Konfiguration

```
su root
dcm console web
service apache2 reload
exit
```

Die Anwendung ist nun verfügbar unter https://geoportal.lkros.de/lkrosmap

2.2 Erforderliche externe Bibliotheken

Folgende externen Bibliotheken werden verwendet und werden in js/app.js geladen:

- jQuery-1.12.0/jquery-1.12.0.min.js
- font-awesome-4.7.0/css/font-awesome.min.css
- OpenLayers/v3.8.2/build/ol-debug.js
- https://openlayers.org/en/v3.20.1/css/ol.css
- proj4js/proj4.js

2.3 osm2po

Wenn auf dem eigenem Server ein Routing-Dienst laufen soll, ist dieser vorher zu installieren und dessen Pfad dann in der Datei osm2poserviceproxy.php einzutragen. Zur Installation siehe: http://osm2po.de/

3 Konfiguration der Anwendung

3.1 3rdparty

In der Datei js/app.js werden die Daten von einem lokalen Verzeichnis geholt, welches in der Variable LkRosMap.path3rdParty hinterlegt ist. Die 3rdparty Dateien können über das Repository https://github.com/pkorduan/kvwmap-server.git aus dem Ordner www/apps/3rdparty entnommen werden.

Am besten man legt den Ordner unter das Verzeichnis /var/docker/www/apps und macht es für Apache mit dem Alias 3rdparty verfügbar, falls noch nicht geschehen.

3.2 osm2poserviceproxy.php

In der Variable $service_url wird der Link zum Service osm2po angegeben. Default ist eine Installation von gdi-service.de. Dieser Dienst kann aber auch lokal auf dem eigenen Server eingerichtet sein. Dann muss die entsprechende Adresse angegeben werden.

4 Hinzufügen eines neuen Layers

Jeder Vektor-Layer holt sich seine Daten aus einem Store. Der Store greift auf eine JSON-Datei zu um die Daten zu laden. Diese JSON-Datei wird mit dem Programm wfs2json erzeugt. wfs2json liest einen WFS und erzeugt daraus das benötigte JSON-File. In der Anwendung lkrosmap wird die URL zu der JSON-Datei für jeden Layer angegeben.

Darüberhinaus wird für jeden Layer ein Modell benötigt.

4.1 WFS Dienst

Der WFS-Dienst muss seine Geometrie in einem Attribut msGeometry oder the_geom liefern. msGeometry wird immer als Punktgeometrie interpretiert. the_geom kann vom Typ Point oder MultiPolygon sein. Je nach dem Typ werden die Features als Punkte oder Flächen dargestellt.

4.2 wfs2json Einrichten

Um eine neue Datei erzeugen zu können, ist eine neue Konfigurationsdatei im Ordner conf anzulegen. Am besten kopiert man sich eine vorhandene Datei unter einem neuen Namen, z.B. LRO_Gemeinden.ini

Abbildung 3: Konfigurationsdateien im conf Verzeichnis von wfs2json

In der INI-Datei gibt es einen Bereich mit Angaben zum abzufragenden WFS [wfs] und einen für die zu erzeugende JSON-Datei [json].

```
[wfs]
url = "https://geoportal.lkros.de/dienste/gebietsgrenzen/wfs?"
featureType = "LRO_Gemeinden"
ns = "ms"
localPath = "wfs/"
webPath = "wfs/"
fileName = "lkrosmap_LRO_Gemeinden.gml"

[json]
localPath = "json/"
webPath = "json/"
fileName = "lkrosmap_LRO_Gemeinden.json"
mandatoryAttribute = "gemeinde"
```

Abbildung 4: Konfigurationsdatei für wfs2json

Das Attribut mandarotyAttribut gibt vor, dass nur Features in die JSON-Datei geschrieben werden, die einen Wert in diesem Attribut haben. Möchte man alle Features haben ohne Pflichtattribut, wird ein Leerer Text als mandatoryAttribute angegeben.

Die JSON-Datei wird nun durch folgenden Aufruf erzeugt:

http://gdi-service.de/wfs2json/?c=LRO_Gemeinde

Im Parameter c wird der Name der INI-Datei angegeben.

Hello, welcome to the WFS to JSON Converter.

Content of ini file.

```
Array
(
    [wfs] => Array
        (
            [url] => https://geoportal.lkros.de/dienste/gebietsgrenzen/wfs?
            [featureType] => LRO_Gemeinden
            [ns] => ms
            [localPath] => wfs/
            [webPath] => wfs/
            [fileName] => lkrosmap_LRO_Gemeinden.gml
        )

    [json] => Array
        (
            [localPath] => json/
            [webPath] => json/
            [fileName] => lkrosmap_LRO_Gemeinden.json
            [mandatoryAttribute] => amt
        )

)

You can specify the configuration file in Parameter c (e.g. c=constant). The programm append the file extension .ini on the give
```

WFS Online-Resource

Read the Capabilities here.

Request WFS

Read GML-Data from WFS with URL
https://geoportal.lkros.de/dienste/gebietsgrenzen/wfs?SERVICE=WFS&REQUEST=GetFeature&VERSION=1.0.0&TYPENAME=LRO_Gemeinden
and store the content on the servers filesystem at
/var/www/apps/wfs2json/wfs/lkrosmap_LRO_Gemeinden.gml
This file is now available for download here.

Convert and Save JSON

Lese Datei /var/www/apps/wfs2json/wfs/lkrosmap_LRO_Gemeinden.gml,
und speicher die konvertierte JSON Datei /var/www/apps/wfs2json/json/lkrosmap_LRO_Gemeinden.json
This file is now available for download here.

---

geometry['type']: MultiPolygon
geometry['coordinate']: [296965.493 5998741.669, 296962.694 5998738.841, 296939.762 5998719.701, 296922.433 5998708.737, 296894.589 5998691.1
296817.938 5998651.075, 296784.023 5998628.041, 296748.585 5998616.987, 296705.603 5998603.564, 296703.121 5998
ogc_fid = 28549
regionalschluessel = 130720211006
schluesselgesamt = 13072006
land = 13
landname = Mecklenburg-Vorpommern
regierungsbezirk = 0
kreis = 72
kreisname = Landkreis Rostock
amt = 211
amtsname = Bad Doberan, Stadt
gemeinde = 6
gemeindename = Bad Doberan, Stadt
beginnt = 2015-01-12T14:07:51Z
endet =
anz_gemarkg = 4
teilflaeche_soll = 1
flaeche = 32858051
weist_auf = DEMVAL72Z000000b

Abbildung 5: Ausgabe der JSON-Dateierzeugung

Im Ergebnis erhält man einen Link zum Capabilities-Dokument, einen zum heruntergeladenen GML-

Dokument und ein Link zur erzeugten JSON-Datei.

Anschließend werden alle Attribute aller Feature zur Ansicht ausgegeben. Diese Daten benötigen wir im nächsten Schritt, wenn das Modell erzeugt wird.

4.3 Modell erzeugen

Das Modell wird im Verzeichnis js/models der Anwendung lkrosmap erzeugt. Am besten man kopiert sich ein vorhandenes Modell unter dem Namen den das Modell tragen soll.

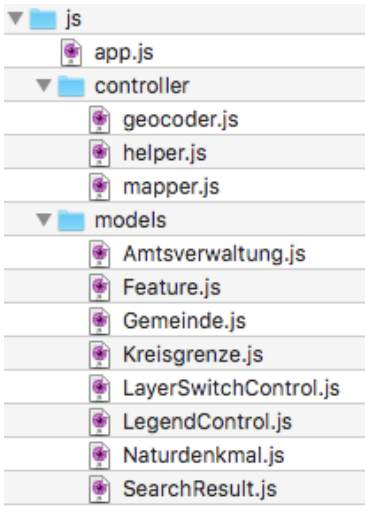

Abbildung 6: Modelle im Verzeichnis models der Anwendung lkrosmap

Diese Datei wird wie folgt bearbeitet:

Zunächst ist der Name der Klasse umzubenennen in den Namen den das Modell tragen soll, z.B. Gemeinden.

```
LkRosMap.models.Gemeinde = function(store) {
  var params = {
    gid: store.ogc_fid,
    type: 'MultiPolygonFeature',
    regionalschluessel: store.regionalschluessel,
    kreisname: store.kreisname,
    kreis: store.kreis,
    amtsname: store.amtsname,
    amt: store.amt,
    gemeinde: store.gemeinde,
    gemeindename: store.gemeindename,
    anz_gemarkg: store.anz_gemarkg,
    flaeche: store.flaeche,
    geometry: new ol.geom.Polygon(store.geometry.coordinates),
    classItem: 'type',
    classes: [{
      name: '',
      expression: function(value) {
        return true
      },
      style: new ol.style.Style({
        stroke: new ol.style.Stroke({
          color: 'rgb(255, 0, 0)',
          width: 1
        })
      }),
      icon: 'Gemeinde'
    }]
  },
```

Abbildung 7: Parameterzuordnung im Modell

Anschließend erfolgt die Zuordnung der Parameternamen zu den Namen der Attribute in der JSON-Datei. Dazu ist die Darstellung der Attribute in der JSON-Datei, die nach der JSON-Dateierzeugung angezeigt wird hilfreich, siehe Abbildung 5 unten.

In unserem Beispiel werden z.b. die Attribute gemeinde und gemeindename hinzugefügt und das Attribut anz_gemeinden durch anz_gemarkg ersetzt.

Des Weiteren werden der Style, z.b. eine andere Farbe und der Name für das Icon gesetzt. Aus dem Namen des Icons wird für Flächenobjekte der Dateiname für das Symbolbild in der Legende erzeugt. Hier im Beispiel wird ein Bild im Ordner img/Gemeinde.png für die Darstellung des Layers in der Legende verwendet, siehe Parameter icon in Abbildung 7.

Die Namen der Parameter können frei gewählt werden. Die Namen der Attribute des Stores richten sich nach den Bezeichnungen aus dem JSON-File. Diese kommen aus der WFS-Definition. Man kann also hier im Modell ein Mapping zwischen den WFS, bzw. JSON-Features und den in der Anwendung lkrosmap darzustellenden Features festlegen. Im Folgenden werden die Parameter für die Sachdatenanzeige in der Methode dataFormatter verwendet.

```
feature.dataFormatter = function() {
  var lines = [];

  lines.push('<b>Name:</b> ' + this.get('gemeindename') + ' (' + this.get('gemeinde') + ')');
  lines.push('<b>Regionalschlüssel:</b> ' + this.get('regionalschluessel'));
  lines.push('<b>im Kreis:</b> ' + this.get('kreisname') + ' (' + this.get('kreis') + ')');
  lines.push('<b>im Amt:</b> ' + this.get('amtsname') + ' (' + this.get('amt') + ')');
  lines.push('<b>Anzahl Gemarkungen:</b> ' + this.get('anz_gemarkg'));
  lines.push('<b>Fläche [km2]:</b> ' + this.get('flaeche'));
  return lines.join('<br>');
};
```

Abbildung 8: Festlegung der Info-Fensterausgabe im dataFormatter

Die Funktion dataFormatter des Models erzeugt den Text, der im Info-Fenster ausgegeben wird. Der Text kann beliebiges HTML enthalten und es kann auf alle Modellparameter, die oben definiert wurden zurückgegriffen werden, siehe Beispiel in Abbildung 8.

```
feature.prepareInfoWindow = function() {
    $('#LkRosMap\\.infoWindowTitle').html('Naturdenkmal');
    $('#LkRosMap\\.infoWindowData').html(this.dataFormatter());
    $('#LkRosMap\\.infoWindowRemoveFeature').hide();
};
```

Abbildung 9: Festlegung der Überschrift im Info-Fenster

In der Funktion prepareInfoWindo muss lediglich der Parameter infoWindowTitle angegeben werden, siehe Abbildung 9, die anderen Angaben sind optional.

Es gibt weitere Funktionen zur Gestaltung der Ausgaben:

titleFormatter … Gestaltet die Ausgabe des Titels.

addressText … Gestaltet die Ausgabe eines Textes, welche den Ort beschreibt. Dieser Text wird in das entsprechende Input-Feld der Routing-Funktion übernommen, wenn von einem Feature aus im Info-Fenster „Route von hier“ oder „Route nach hier“ gewählt wird.

4.4 Modell im Controller registrieren

Derzeit muss im mapper Controller in der Funktion loadFeatures der Case für das neue Modell noch manuell hinzugefügt werden, siehe Abbildung 11.

Der Mapper-Controller befindet sich in der Datei js/controller/mapper.js, siehe Abbildung 6.

4.5 Layer in Startseite Eintragen

Nun sind alle Voraussetzungen zur Definition des Layers erfüllt und der Layer kann jetzt in eine Startseite eingetragen werden.

Abbildung 10: Startseiten im apps Verzeichnis von lkrosmap

Die Startseiten befinden sich im apps Verzeichnis der Anwendung lkrosmap, siehe Abbildung 10. In unserem Beispiel wird der Layer Gemeinden zur Startseite Gebietsgrenzen.html hinzugefügt, siehe Abbildung 11.

Der Konfigurationsblock für einen Layer enthält den Namen, die Quellenangabe für die JSON-Datei, die Modellbezeichnung und einen Text für das Copy-Right.

Die Layer werden in der Reihenfolge ihrer Eintragung in der Datei gezeichnet.

```
loadFeatures: function(store, layer, model) {
  var source = layer.getSource(),
      i;

  for (i = 0; i < store.length; i++) {
    switch (model) {
      case 'Naturdenkmal': {
        feature = new LkRosMap.models.Naturdenkmal(store[i]);
      } break;

      case 'Kreisgrenze': {
        feature = new LkRosMap.models.Kreisgrenze(store[i]);
      } break;

      case 'Amtsverwaltung': {
        feature = new LkRosMap.models.Amtsverwaltung(store[i]);
      } break;

      case 'Gemeinde': {
        feature = new LkRosMap.models.Gemeinde(store[i]);
      } break;

      default: {
        store[i].icon = 'Default';
        feature = new LkRosMap.models.Feature(store[i]);
      }
    }
    source.addFeature(feature);
  }
  return layer;
},
```

Abbildung 11: Cases für Modell in loadFeature Funktion des mapper Controller

Wenn die Startdatei nun mit folgendem Link aufgerufen wird, erscheint die Karte mit den eingebundenen Layern, siehe Abbildung 12.

http://gdi-service.de/lkrosmap/apps/Gebietsgrenzen.html

```
layers: [{
    index: 0,
    name: 'Gemeinden',
    url: '../../wfs2json/json/lkrosmap_LRO_Gemeinden.json',
    model: 'Gemeinde',
    attribution: 'Gemeinden Mecklenburg-Vorpommern<br>'
}, {
    index: 1,
    name: 'Amtsverwaltungen',
    url: '../../wfs2json/json/lkrosmap_LRO_Gemeindeverbaende.json',
    model: 'Amtsverwaltung',
    attribution: 'Amtsverwaltungen Mecklenburg-Vorpommern<br>'
}, {
    index: 2,
    name: 'Kreisgrenzen',
    url: '../../wfs2json/json/lkrosmap_LRO_Kreise.json',
    model: 'Kreisgrenze',
    attribution: 'Landkreise Mecklenburg-Vorpommern<br>'
}],
center: [12.21, 53.91],
```

Abbildung 12: Konfiguration der Layer in einer Startseite

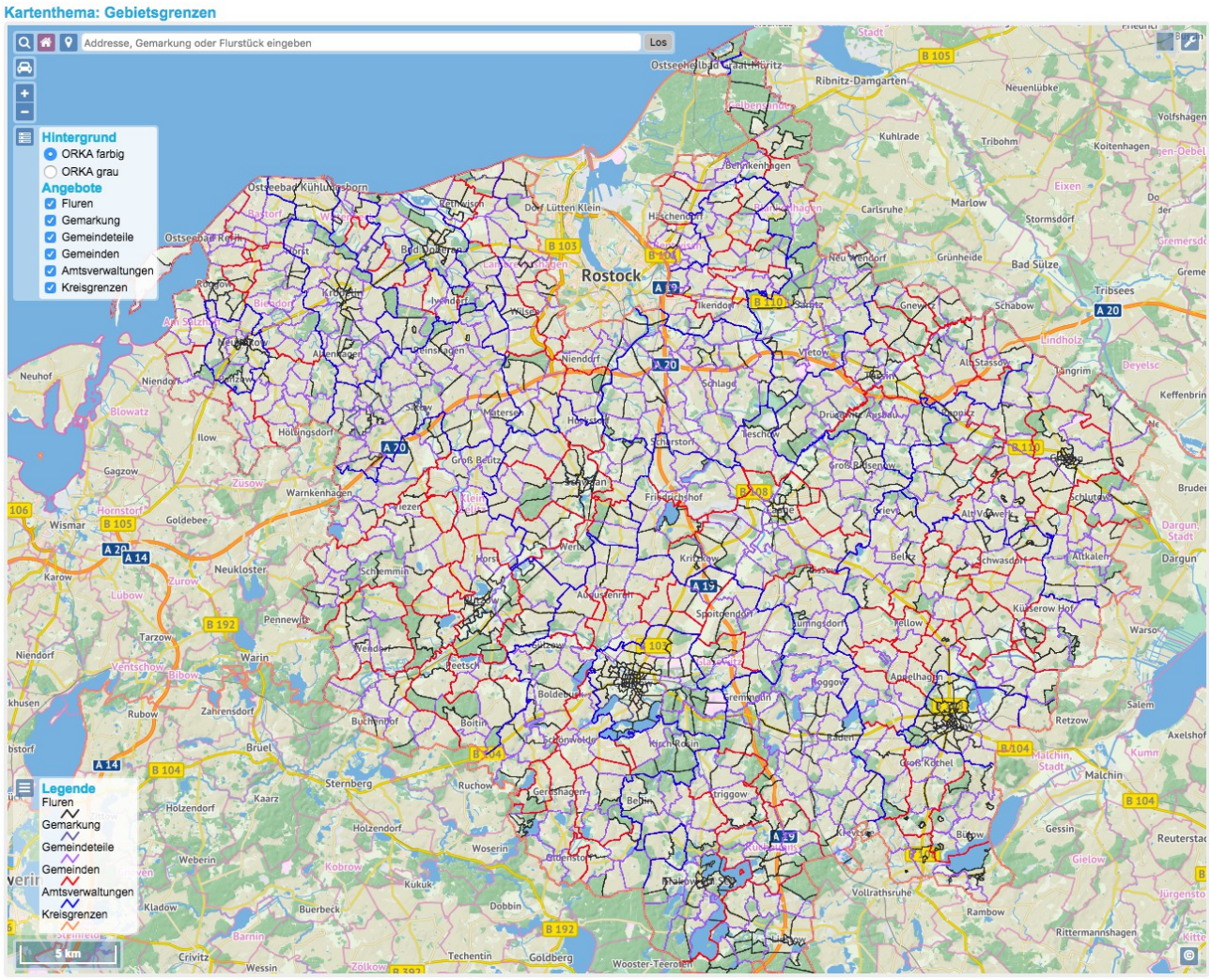

Abbildung 13: Darstellung der Layer im LayerSwitch- und Legenden-Control

Abbildungsverzeichnis

Tabellenverzeichnis