Ajuntament de Barcelona

Diseña mapas interactivos con OpenLayers.

Día 4 de 4, Modulo 2h, 2021

Barcelonactiva

Barcelona Activa: Qui som?

Barcelona Activa, integrada en l'àrea d'Economia, Empresa i Ocupació, és l'organització executora de les polítiques de promoció econòmica de l'Ajuntament de Barcelona.

Des de fa 25 anys impulsa el creixement econòmic de Barcelona i el seu àmbit d'influència donant suport a les empreses, la iniciativa emprenedora i l'ocupació, alhora que promociona la ciutat internacionalment i els seus sectors estratègics; en clau de proximitat al territori.

Barcelona Activa va ser guanyadora del Gran Premi del Jurat 2011, atorgat per la DG d'Empresa i Indústria de la Comissió Europea en el marc dels *European Enterprise Awards*, per la iniciativa empresarial més creativa i inspiradora d'Europa.

Àrees d'activitat de Barcelona Activa

Barcelona Activa s'estructura en tres grans blocs de serveis a les **Empreses**, a l'**Emprenedoria** i a la **Ocupació**. La **Formació** és un instrument transversal present en els tres blocs, així com també tot el relacionat amb l'economia social.

Una xarxa d'Equipaments Especialitzats

Xarxa de Proximitat

13 antenes Cibernàrium a biblioteques
10 punts d'atenció en Ocupació

Índice

- Día 1: Conceptos básicos y primeros mapas con capas ráster
- Día 2: Capas vectoriales de GeoJSON y TopoJSON y su diseño
- Día 3: Controles, estilos dinámicos, eventos, animaciones y capas WMS
- **Día 4: Overlays, proyecciones, plugins y teselas vectoriales**

Índice día 4

- Overlays
- Proyecciones
- Teselas vectoriales
- Plugins
- Build para entornos de producción
- QGIS Server como servidor de mapas

Capas vectoriales de datos: Popup

Hay diferentes técnicas para mostrar popups encima de un mapa OpenLayers, la que me resulta más cómoda y flexible es utilizando este elemento HTML con estilos definidos en el fichero *popup.css*:

```
<div id="popup" class="ol-popup">
  <a href="#" id="popup-closer" class="ol-popup-closer"></a>
  <div id="popup-content"></div>
</div>
```

Overlay es una clase de OpenLayers que muestra un contenido encima del mapa ligado a una coordinada:

```
var overlayPopup = new ol.Overlay({
  element: document.getElementById('popup'),
  autoPan: true,
  autoPanAnimation: {
    duration: 250
  }
});
map.addOverlay(overlayPopup);
```

Capas vectoriales de datos: Popup

Asignamos el contenido al popup y usamos la función *setPosition* de *Overlay* para posicionar el popup sobre la coordinada clickada:

```
map.on('click', function(evt) {
  let coordinate = evt.coordinate;
  let pixel = map.getPixelFromCoordinate(coordinate);
  let output = '';

  overlayPopup.setPosition(undefined);

  map.forEachFeatureAtPixel(pixel, function(feature) {

    if (feature.get('name') === undefined) {
      output += 'Capital: ' + feature.get('city') + '<br>';
    }
    else {
      output += feature.getProperties().name + '<br>';
    }

    document.getElementById('popup-content').innerHTML = output;
    overlayPopup.setPosition(coordinate);
  });
});
```

Capas vectoriales de datos: Popup

El popup puede contener cualquier código HTML, como ejemplo añadimos la bandera de cada país:

```
map.forEachFeatureAtPixel(pixel, function(feature) {

  if (feature.get('name') === undefined) {
    output += 'Capital: ' + feature.get('city') + '<br>';
    output += '<img style="width:100px"
src="https://flagpedia.net/data/flags/w580/' + feature.get('tld') + '.png"
/><br>';
  }
  else {
    output += 'Country: ' + feature.getProperties().name + '<br>';

  }

  document.getElementById('popup-content').innerHTML = output;
  overlayPopup.setPosition(coordinate);
});
```

Proyecciones

OpenLayers viene con funcionalidades para utilizar transformaciones entre diferentes proyecciones basado en proj4. Para utilizar otras proyecciones tenemos que incluir la librería proj4js, que permite utilizar cualquier proyección de este proyecto en la librería.

Las únicas proyecciones que vienen a bordo son la EPSG:3857 (por definición) y la EPSG:4326 (WSG84). Para aplicarla usamos el parámetro *projection* de la clase *View*:

```
view: new View({
  projection: 'EPSG:4326',
  center: [0, 0],
  zoom: 2,
}),
```

Proyecciones

Para cualquier otra proyección, primero consultamos su definición – en este ejemplo EPSG:25831 - y la definimos usando la clase *Projection*. A partir de ahí se puede utilizar con el parámetro *projection* en la *View* (ejemplo *barcelona_proyecciones.html*):

```
proj4.defs(
  "EPSG:25831", "+proj=utm +zone=31 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0
+units=m +no_defs"
);
ol.proj.proj4.register(proj4);

const projection = new ol.proj.Projection({
  code: 'EPSG:25831',
  units: 'm',
});

view: new ol.View({
  center: ol.proj.fromLonLat([ 2.027665411390514, 41.36152437117059 ]),
  zoom: 18,
  projection: projection
})
```

Proyecciones

Aquí hay un buen tutorial para explicar en profundidad las reproyecciones.

Ejemplos:

- Raster reprojection demo
- Reprojection with EPSG.io database search
- Image reprojection

Teselas vectoriales

De la misma manera que podamos cargar teselas ráster de una pirámide de teselas, también podemos hacerlo con teselas vectoriales. Al contrario de cargar datos vectoriales de un fichero para todo el mapa y todas las resoluciones, podemos cargar la información ajustada a la vista actual. Eso es necesario en el caso de cartografía base o también capas adicionales que tengan mucha información y por lo tanto pesaría demasiado cargarlo en un fichero único.

Mapbox Vector Tiles (MVT) están en un formato binario muy eficiente, además tiene asociado una especificación de estilos llamada Mapbox Styles. Con esta combinación podemos cargar los datos vectoriales al navegador y aplicarles un estilo en el momento usando JavaScript. En seguida miraremos como utilizar los MVT y sus estilos desde OpenLayers

Teselas vectoriales

Usamos la clase VectorTile para cargar teselas vectoriales. Como fuente utilizamos datos de un mapa callejero de OpenStreetmap. El proveedor permite acceso libre por fines de prueba, pero exige una llave. El ejemplo *vector-tiles.html* contiene una llave, para ejemplos propios haría falta crearse una cuenta en https://cloud.maptiler.com/ y usar la llave personal.

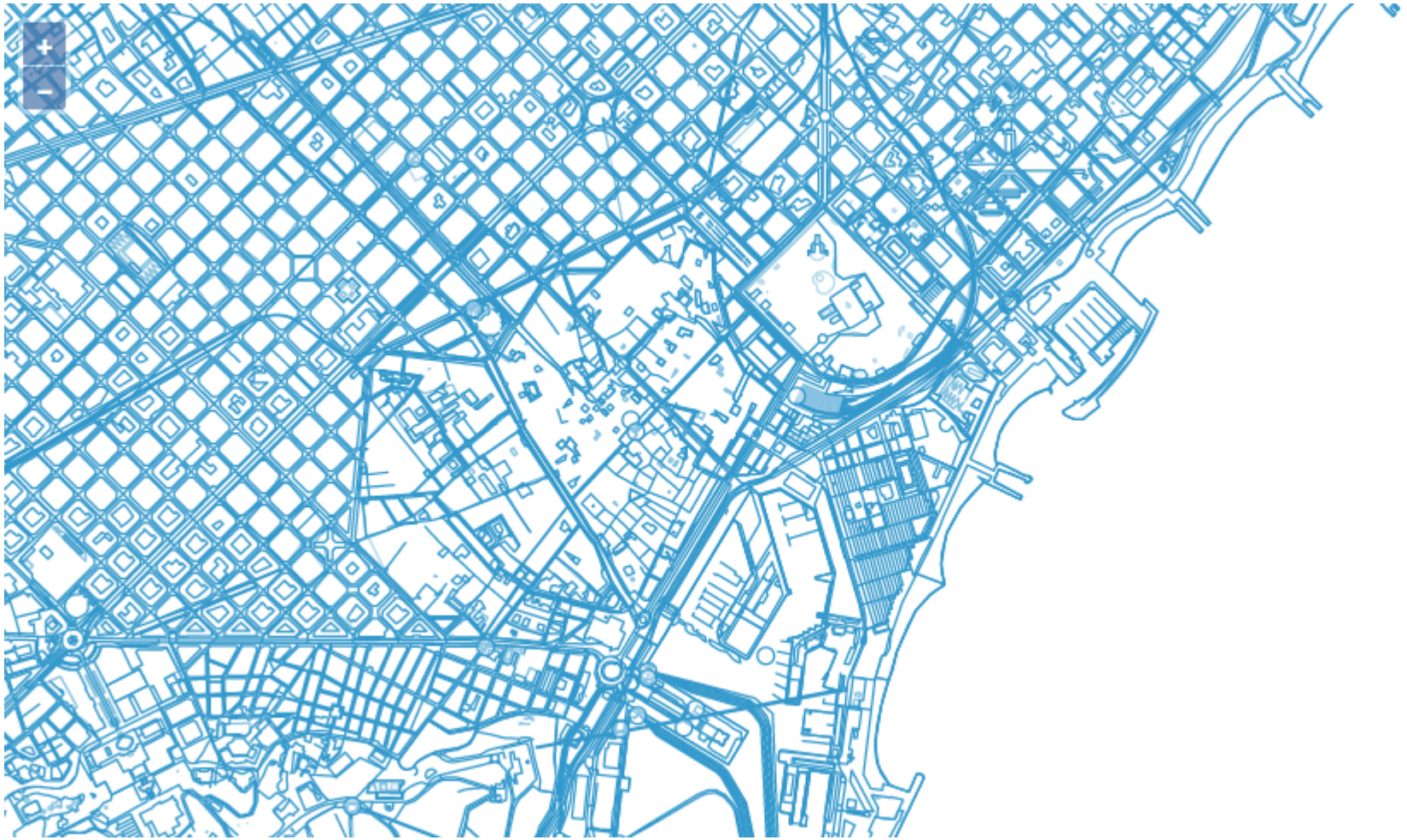
```
const layer = new ol.layer.VectorTile({

  source: new ol.source.VectorTile({

    attributions: [
      '<a href="https://www.maptiler.com/copyright/">&copy; MapTiler</a>',
      '<a href="https://www.openstreetmap.org/copyright">&copy; OpenStreetMap
contributors</a>'
    ],
    format: new ol.format.MVT(),
    url: 'https://api.maptiler.com/tiles/v3/{z}/{x}/{y}.pbf?key='+key,
    maxZoom: 14
  })
});

map.addLayer(layer);
```

Teselas vectoriales



Teselas vectoriales

Capas vectoriales siempre necesitan estilos para que se vean aceptables. Mapbox ofrece un editor visual con muy buena usabilidad llamado Mapbox Studio, Maputnik es una buena alternativa de software libre.

El paquete ol-mapbox-style añade soporte para los estilos de Mapbox Style a OpenLayers. Una vez cargado este paquete, podemos aplicar estilos predefinidos con la siguiente orden:

```
var styleJson = 'https://api.maptiler.com/maps/bright/style.json?key='+key;
olms.apply(map, styleJson);
```

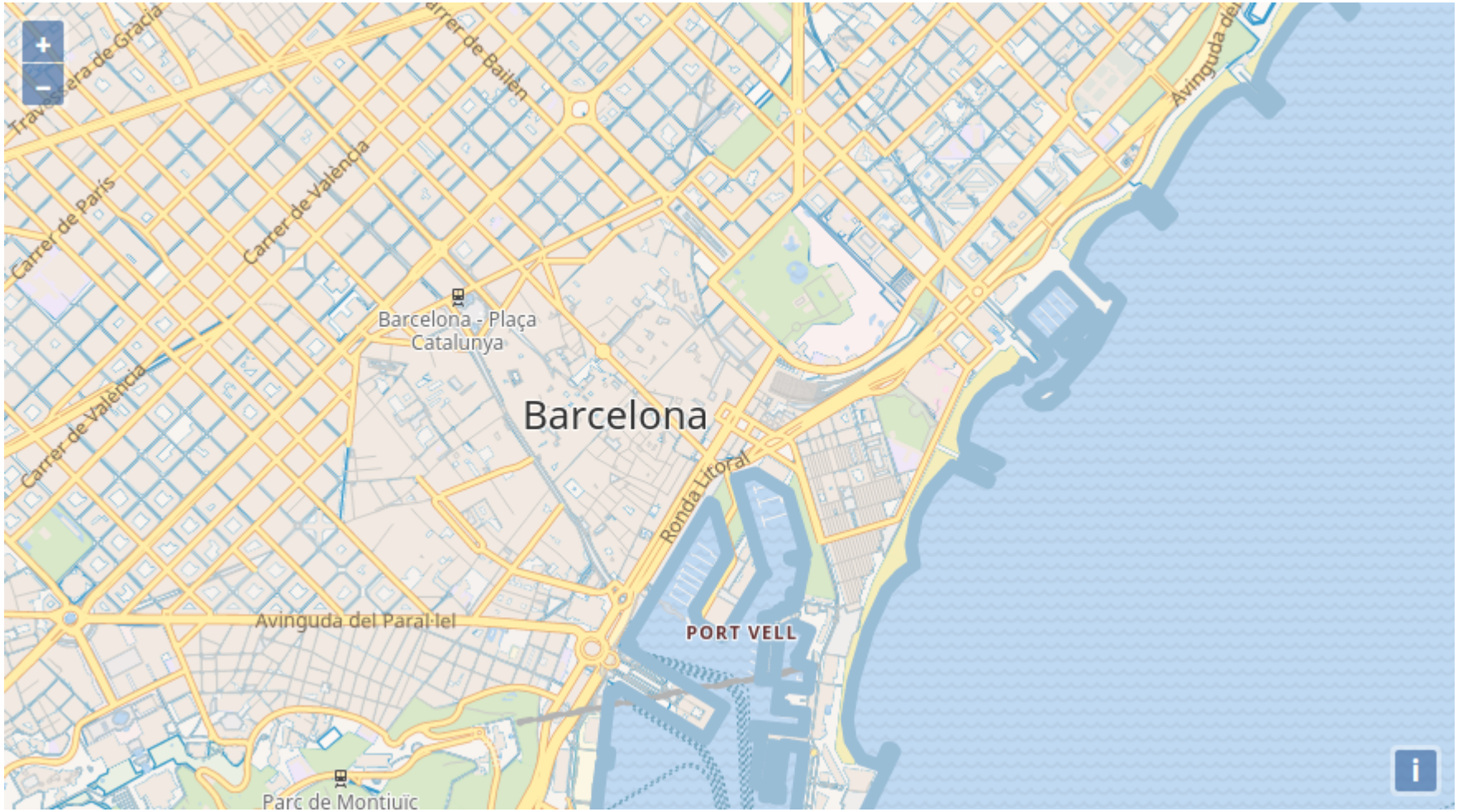
Teselas vectoriales

Capas vectoriales siempre necesitan estilos para que se vean aceptables. Mapbox ofrece un editor visual con muy buena usabilidad llamado Mapbox Studio, Maputnik es una buena alternativa de software libre.

El paquete ol-mapbox-style añade soporte para los estilos de Mapbox Style a OpenLayers. Una vez cargado este paquete, podemos aplicar estilos predefinidos con la siguiente orden:

```
var styleJson = 'https://api.maptiler.com/maps/bright/style.json?key='+key;
olms.apply(map, styleJson);
```

Teselas vectoriales



Teselas vectoriales

Una característica de las capas vectoriales es que siempre podemos acceder los datos, pero hay que tener en cuenta que los MVT están optimizados para el rendering y no contienen mucha información adicional. En todo caso podemos acceder esta información, en este ejemplo dibujamos un rectángulo sobre los features seleccionados:

```
map.on('pointermove', function(event) {

  source.clear();
  map.forEachFeatureAtPixel(event.pixel, function(feature) {

    //console.log(feature.getProperties());
    const geometry = feature.getGeometry();

    source.addFeature(new
ol.Feature(ol.geom.Polygon.fromExtent(geometry.getExtent())));
  }, {
    hitTolerance: 2
  });
});
```

Teselas vectoriales

Teselas vectoriales

Producir y servir vector tiles de forma estática:

1. Preparar mapa en QGIS
2. Exportar capa en GeoJSON
3. Convertirlo en mbtiles utilzando https://github.com/mapbox/tippecanoe
4. Servir pirámide de teselas (mbtiles) con servidor de teselas, por ejemplo [Tileserver](https://github.com/maptiler/tileserver-gl) (o sistema en la nube como https://www.maptiler.com/)
5. Usar MVT como VectorTile en OpenLayer

Producir y servir vector tiles de dinámica (live desde BDD):

Usar un servidor de tiles tipo Tegola o similar -> https://github.com/mapbox/awesome-vector-tiles

Plugins

Podemos extender la funcionalidad de OpenLayers usando diferentes plugins. Aquí algunas recomendaciones de la lista oficial:

- Layer Switcher
- GeoCoder
- sidebar-v2
- ol-mapbox-style
- ol-ext

Otra herramienta útil:

- Geostyler: Traducción entre diferentes formatos como SLD, OpenLayers, QGIS, Mapbox

Plugins: LayerSwitcher

Basado en el ejemplo del día 3 barcelona-bibliotecas.html creamos un ejemplo usando el plugin LayerSwitcher. Este plugin nos permite añadir un selector de capas sin demasiada programación.

El primer paso es incluir la librería (tanto JS como CSS!):

```
<script src="https://cdn.jsdelivr.net/npm/ol-layerswitcher@3.8.3/dist/ol-layerswitcher.min.js"></script>
```

Después hace falta dar a cada capa que queremos hacer aparecer un nombre usando el parámetro *title*. Además las capas base necesitan el parámetro *type*. Asé quedará la definición para una cape base:

```
new ol.layer.Tile({
  title: 'OpenStreetMap',
  type: 'base',
  source: new ol.source.OSM()
}),
```

Plugins: LayerSwitcher

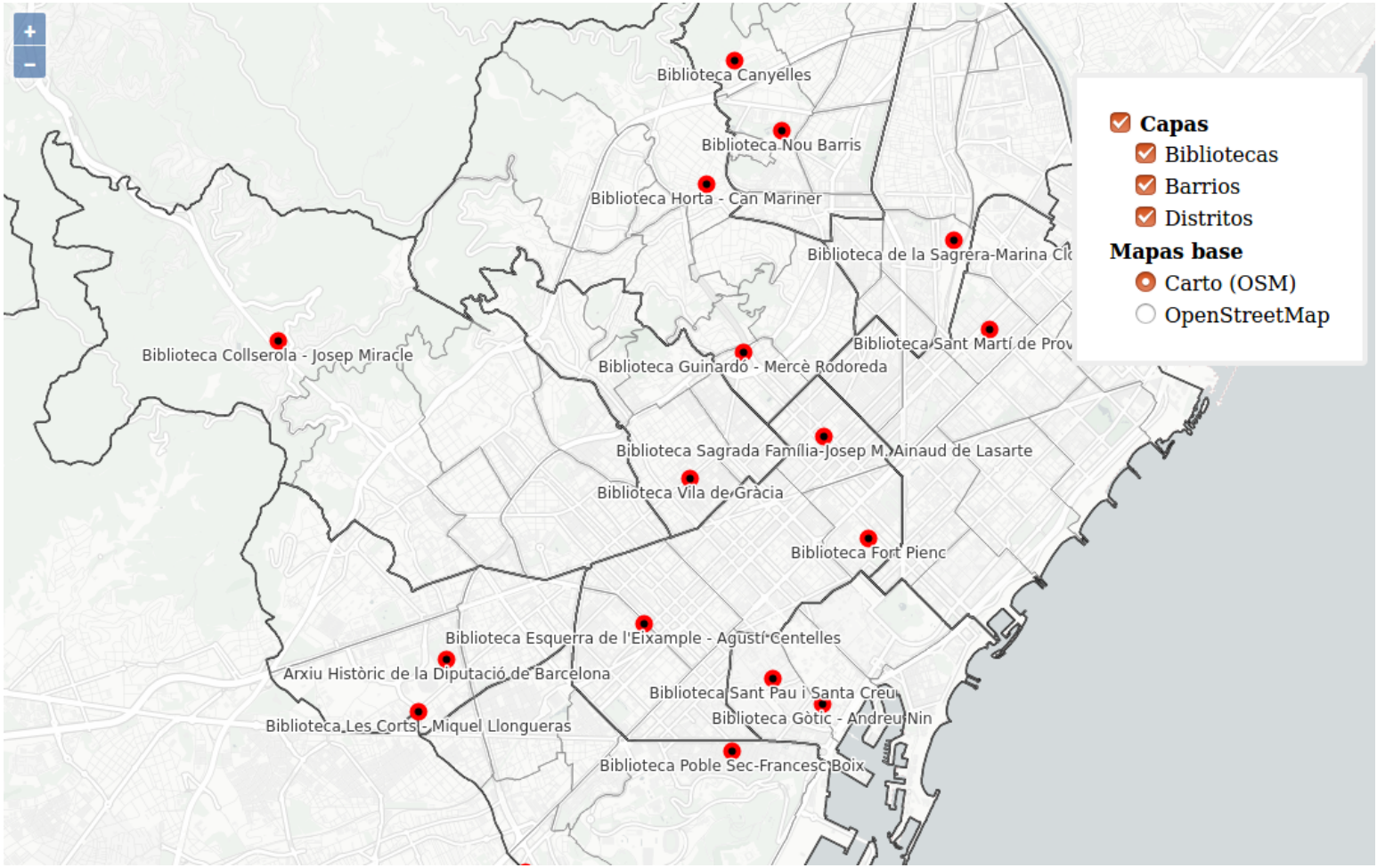
En el siguiente paso creamos una instancia del LayerSwitcher:

```
var layerSwitcher = new LayerSwitcher();
map.addControl(layerSwitcher);
```

A partir de este momento ya debería aparecer el selector de capas, lo podemos refinar más añadiendo parámetros y agrupando elementos usando la clase *Group*:

```
new ol.layer.Group({
  title: 'Mapas base',
  layers: [
    ...
  ]
}),
```

Plugins: LayerSwitcher



Build para entornos de producción

Aplicaciones modernas de JavaScript funcionan mejor cuando se usa y se edita módulos. El modo recomandado de utilizar OpenLayers es instalando el paquete ol. Este tutorial de mostrará paso por paso como montar un entorno de desarrollo basado en node. Una vez montado este entorno un mapa base de OpenLayers tendrá este aspecto:

```
import 'ol/ol.css';
import {Map, View} from 'ol';
import TileLayer from 'ol/layer/Tile';
import OSM from 'ol/source/OSM';

const map = new Map({
  target: 'map',
  layers: [
    new TileLayer({
      source: new OSM()
    })
  ],
  view: new View({
    center: [0, 0],
    zoom: 0
  })
});
```

QGIS Server

QGIS Server es un servidor GIS de software libre que sirve WMS, WFS, etc. utilizando las mismas librerías de rendering que QGIS. Una gran ventaja de utilizar QGIS Server como servidor de mapas es que los mapas publicados en web tienen exactamente el mismo aspecto que los de escritorio. Por lo tanto es una forma eficiente de llevar tus mapas de escritorio a web.

En la documentación oficial de QGIS Server se explican las diferentes opciones de instalación, yo tengo una guía básica con mis apuntes para instalar una versión de prueba. En todo caso cabe mencionar que se necesito un servidor con ciertas capacidades y adicionalmente un servidor web y un base de datos PostgreSQL. Como sistema de cache recomiendo utilizar Mapproxy.

Aquí un ejemplo en directo: http://mapa.psig.es/ssa/

Barcelonactiva

Ajuntament de Barcelona

bcn.cat/barcelonactiva
bcn.cat/cibernarium