



Ajuntament  
de Barcelona

# Diseña mapas interactivos con OpenLayers.

Día 3 de 4, 2022



# 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.

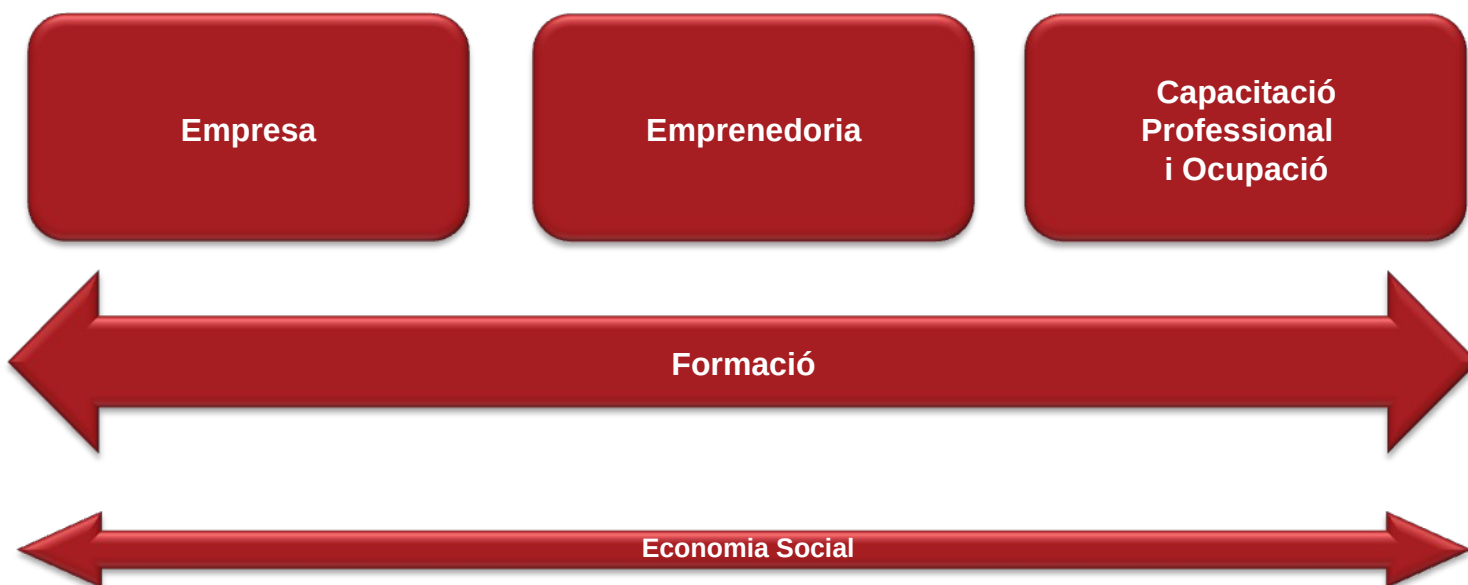


Barcelona Activa va ser guanyadora del Gran Premi del Jurat 2011, atorgat per la DG d'Empresa i Indústria de la Comissió Europea en el marc dels *European Enterprise Awards*, per la iniciativa empresarial més creativa i inspiradora d'Europa.



# 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



Seu Central



Centre  
Iniciativa Emprenedora



Incubadora  
Glòries



Almogàvers  
Business Factory



Parc Tecnològic  
BCN Nord



Centre  
Desenvolupament  
Professional Porta22



Cibernàrium  
MediaTIC



Convent  
de Sant Agustí



Can Jaumandreu



Ca n'Andalet

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 3

- Controles
- Estilos dinámicos para capas vectoriales de datos
- Eventos y interacciones con capas vectoriales de datos
- Capas WMS
- Animaciones



# Controles

Controles son widgets visibles con un elemento HTML. Al contrario de los Overlays están en una posición fija en la pantalla. OpenLayers por definición viene con 3 controles activados:

- Zoom
- Atribución
- Rotación

Podemos desactivar estos elementos indicando un control en el mapa con el siguiente código:

```
controls: ol.control.defaults({  
  zoom: false,  
  attribution: false,  
  rotate: false  
}),
```

Para aplicar los siguientes códigos, utilizaremos de ejemplo base el fichero *barcelona.html*



# Controles: Ejemplo base barcelona.html

```
var map = new ol.Map({
  target: 'map',
  layers: [
    new ol.layer.Tile({
      source: new ol.source.XYZ({
        url:
'https://{1-4}.basemaps.cartocdn.com/light_nolabels/{z}/{x}/{y}.png',
      })
    }),
    new ol.layer.Vector({
      source: new ol.source.Vector({
        format: new ol.format.TopoJSON(),
        url: 'districtes.json'
      })
    })
  ],
  view: new ol.View({
    center: ol.proj.fromLonLat([2.183333, 41.383333]),
    zoom: 12
  })
});
```





## Controles: Atribución

Cuando se usa fuentes como OSM o Bing Maps, las atribuciones correctas aparecen automáticamente. En el caso de OSM, por ejemplo, se exige indicar la licencia con “[© OpenStreetMap contributors](#)”. Cabe destacar que es obligatorio nombrar correctamente la fuente y aunque técnicamente es posible esconder la atribución para OSM, hace falta mostrarla.

Si usamos como fuente la clase *ol.source.XYZ()* entonces hace falta especificar manualmente la atribución correcta. Como ejemplo añadimos la correcta atribución para las mapas base del [ICGC](#):

```
new ol.layer.Tile({
  source: new ol.source.XYZ({
    url: 'https://tilemaps.icgc.cat/mapfactory/wmts/topo_suau/CAT3857/{z}/{x}/{y}.png',
    attributions: 'Institut Cartogràfic i Geològic de Catalunya CC-BY-SA-3'
  })
})
```



# Controles

OpenLayers además viene con varios controles habituales de mapa que están preconfigurados. Mirando la clase [Control](#) vemos, entre otros, los siguientes subclases:

- Scaleline: Línea de escala
- OverviewMap: Mapa general
- Fullscreen: Pantall completa
- ZoomSlider: Deslizador de zoom
- ZoomToExtent: Zoom a la medida



## Controles: Línea de escala

Para usar uno de estos controles, tenemos que crear una instancia de la clase en cuestión, definir ciertos parámetros y añadirlo al mapa:

```
let scaleLine = new ol.control.ScaleLine({  
  units: 'metric',  
  minWidth: 100  
});  
  
map.addControl(scaleLine);
```

La posición de los controles esta predefinido, para cambiarlo definimos un estilo CSS:

```
.ol-scale-line {  
  right: 8px;  
  left: auto;  
}
```



## Controles: Zoom a la medida

Con este control podemos invocar un zoom a un cierto área del mapa. Definimos el área con la clase *extent*, en nuestro caso usamos coordenadas decimales EPSG:4326 que transformamos en los métricos de EPSG:3857 usado por definición por OpenLayers:

```
let zoomToExtentControl = new ol.control.ZoomToExtent({
  extent: ol.proj.transformExtent([
    2.0537286604003904,
    41.46802692765104,
    2.2432428205566404,
    41.31580776736919
  ],
  'EPSG:4326',
  'EPSG:3857'
});

map.addControl(zoomToExtentControl);
```



## Ejercicio

Define un control del tipo zoom a la media con el mismo extensión de la vista inicial.

Notas:

- Cualquier coordinada o extent se puede definir como variable
- Mira parámetro *extent* de la clase [View](#)

```
let zoomToExtentControl = new ol.control.ZoomToExtent({  
  extent: ol.proj.transformExtent(  
    [  
      2.0537286604003904,  
      41.46802692765104,  
      2.2432428205566404,  
      41.31580776736919  
    ],  
    'EPSG:4326',  
    'EPSG:3857'  
  )  
});  
  
map.addControl(zoomToExtentControl);
```



## Ejercicio solucionado

El fichero *barcelona\_controles\_zoomtoextent.html* contiene una solución. Las partes importantes del código son:

```
let defaultExtent = ol.proj.transformExtent([ 2.0537286604003904,  
41.46802692765104 , 2.2432428205566404, 41.31580776736919 ], 'EPSG:4326',  
'EPSG:3857');  
  
view: new ol.View({  
  center: ol.proj.fromLonLat([2.183333, 41.383333]),  
  zoom: 12,  
  extent: defaultExtent  
});  
  
let zoomToExtentControl = new ol.control.ZoomToExtent({  
  extent: defaultExtent  
});  
  
map.addControl(zoomToExtentControl);
```



# Controles: Mapa general

Para usar uno de estos controles, tenemos que crear una instancia de la clase en cuestión, definir ciertos parámetros y añadirlo al mapa:

```
let overviewMapControl = new ol.control.OverviewMap({  
  layers: [  
    new ol.layer.Tile({  
      source: ol.source.OSM();  
    }),  
  ],  
});  
  
map.addControl(overviewMapControl);
```



# Estilos para capas vectoriales de datos

Ahora utilizaremos información del mismo archivo vectorial para el diseño de una capa.

La clase Feature contiene un elemento del archivo vectorial, en nuestro caso por ejemplo un distrito. Cada *feature* tiene una geometría, que en el caso del distrito es un polígono. Además puede contener propiedades textuales con información sobre el *feature*, por ejemplo el nombre del distrito.

Para sacar estas propiedades usamos la función *feature.get()* usando como parámetro una llave de las propiedades del archivo GeoJSON:

```
feature.get( 'NAME' );
```





# Estilos para capas vectoriales de datos

También podemos sacar toda la información de un *feature* usando:

```
feature.getProperties();
```

Añadimos una función para mostrar la información guardado en la capa.

Primero hay que convertir la coordenada en un pixel, para luego pasarlo a la función *forEachFeatureAtPixel*, que devuelve todos los *features* en el punto cliquado:

```
map.on('click', function(evt) {  
  let pixel = map.getPixelFromCoordinate(evt.coordinate);  
  map.forEachFeatureAtPixel(pixel, function(feature) {  
    console.log(feature.get("NAME"), feature.getProperties());  
  });  
});
```



# Estilos para capas vectoriales de datos

En la clase anterior vimos como aplicar estilos estáticos a capas vectoriales. Aprendimos como aplicar un mismo estilo tipo *Fill* o *Stroke* a toda una capa.

Ahora sacaremos información de los vectores y la aplicaremos dinámicamente a cada *feature*. Para eso necesitamos una **función de estilos** (*Style Function*), que aplica a cada *feature* su estilo propio:

```
style: function(feature, resolution) {  
  let red = (feature.get("POP_RANK")-10)/7*255;  
  
  return new ol.style.Style({  
    fill: new ol.style.Fill({  
      color: "rgb(" + red + ",0,0)"  
    })  
  })  
}
```



# Estilos para capas vectoriales de datos

```
function vectorStyleFunction(feature, resolution) {  
  let r = feature.get("POP2020")/20000*10;  
  
  return new ol.style.Style({  
    image: new ol.style.Circle({  
      stroke: new ol.style.Stroke({  
        width: 2,  
        color: "#000"  
      }),  
      fill: new ol.style.Fill({  
        color: "#fff"  
      }),  
      radius: r  
    })  
  })  
}  
  
...  
  
style: vectorStyleFunction
```



## Ejercicio: Barcelona Bibliotecas

Cargar un fichero GeoJSON con las bibliotecas públicas de Barcelona y aplicar estilos de imagen y texto para que salga un ícono y una etiqueta en la localización de cada biblioteca Usamos como ejemplo base *barcelona.html* y así también vemos en que barrio y distrito se ubica cada biblioteca.



# Estilos para capas vectoriales de datos

Ahora usamos este conocimiento para añadir etiquetas individuales a los elementos vectoriales del mapa:

```
style: function(feature, resolution) {  
  return new ol.style.Style({  
    text: new ol.style.Text({  
      text: feature.get('name'),  
      font: '12px sans-serif',  
      stroke: new ol.style.Stroke({  
        color: '#fff',  
        width: 3  
      })  
    })  
  })  
}
```

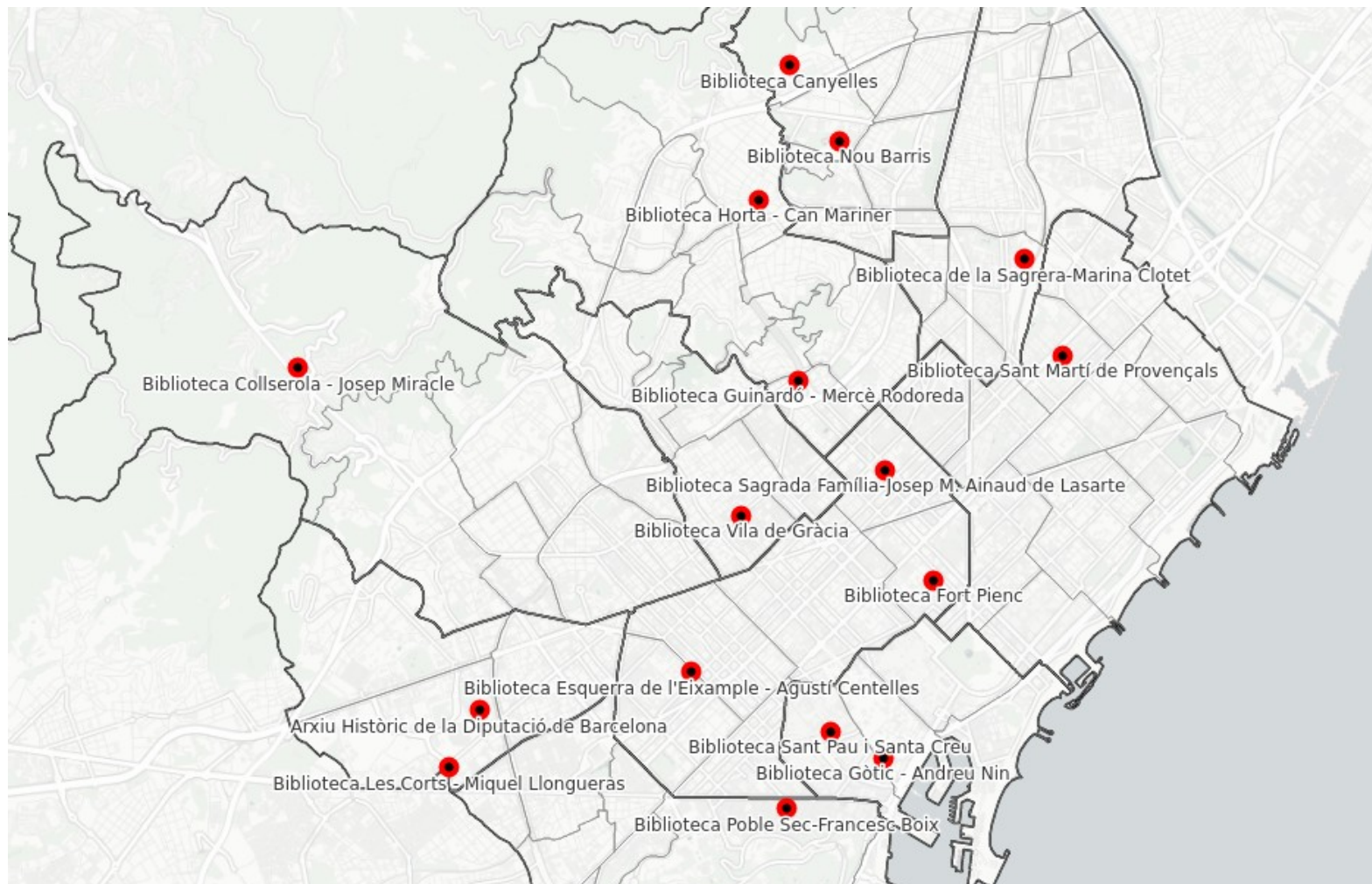


# Estilos para capas vectoriales de datos

```
style: function(feature, resolution) {  
  return new ol.style.Style({  
    image: new ol.style.Circle({  
      stroke: new ol.style.Stroke({  
        width: 4,  
        color: "#ff0000"  
      }),  
      fill: new ol.style.Fill({  
        color: "#000"  
      }),  
      radius: 5  
    }),  
    text: new ol.style.Text({  
      text: feature.get('name'),  
      font: '12px sans-serif',  
      stroke: new ol.style.Stroke({  
        color: '#fff',  
        width: 3  
      }),  
      offsetY: 12  
    })  
  })  
}
```



# Resultado Barcelona Bibliotecas





# Declutter

Para evitar que se superpongan etiquetas una encima de otra, se puede añadir la opción *declutter* a la capa:

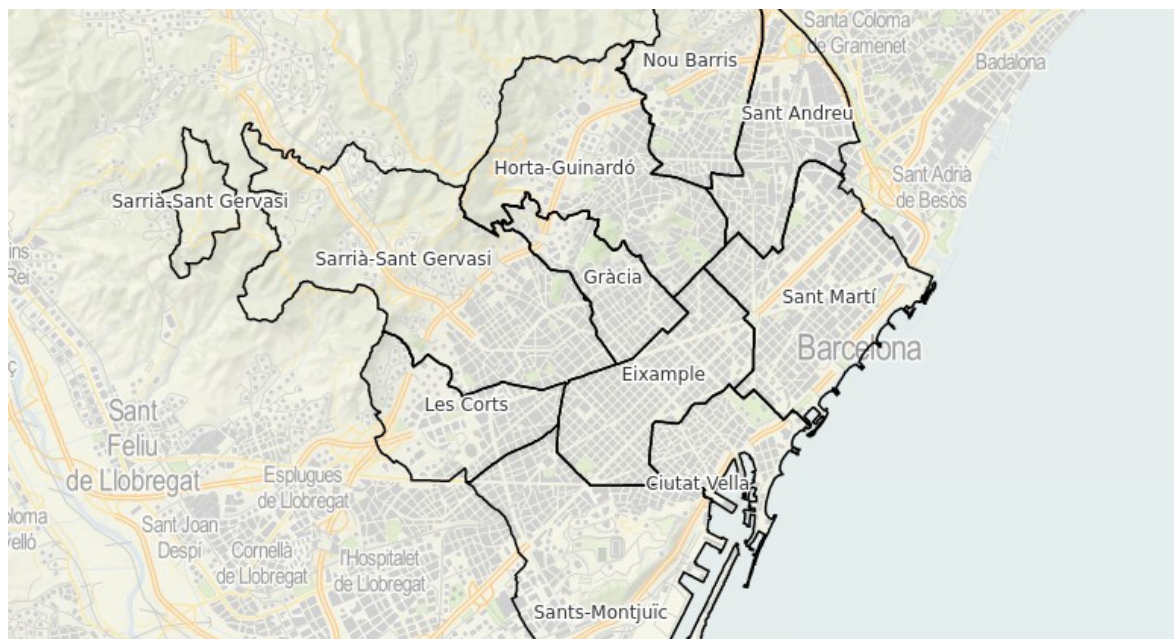
```
new ol.layer.Vector({  
  source: new ol.source.Vector({  
    format: new ol.format.GeoJSON(),  
    url: 'bibliotecas.json'  
  }),  
  style: ...  
  
  declutter: true  
})
```





## Ejercicio: Barcelona Distritos

Usamos el ejemplo inicial *barcelona.html*, borramos las capas barrios y bibliotecas y nos quedamos solamente con la capa de distritos. Después añadimos un estilo de texto para mostrar nombre y ID de cada distrito en el centro de su polígono.





# Ejercicio: Solución Barcelona Distritos

```
style: function(feature, resolution) {  
  return new ol.style.Style({  
    stroke: new ol.style.Stroke({  
      color: '#000',  
      width: 1.5,  
    }),  
  
    text: new ol.style.Text({  
      text: feature.get('NOM'),  
      font: '12px sans-serif',  
      stroke: new ol.style.Stroke({  
        color: '#fff',  
        width: 3  
      }),  
      overflow: true  
    })  
  })  
}
```



## Ejercicio: Bonus Barcelona Distritos

También podemos usar un valor del GeoJSON para crear un mapa tipo choropleth. Aquí una solución muy básica para mapear valores a colores:

```
console.log(feature.get('NOM'), feature.get('AREA'));  
  
let c = 255 - feature.get('AREA') / 300000000 * 255;  
  
// calcular area  
console.log( ol.sphere.getArea(feature.getGeometry() ) );
```



# Interacción con capas vectoriales de datos

Para empezar utilizamos un `<div>` para mostrar la información:

```
<div id="name" style="font-size: 24px;"></div>
```

Definimos un evento *pointermove* que reacciona a cualquier movimiento del ratón y lo muestra en el `<div>` abajo del mapa (*capa\_vector\_interaccion.html*):

```
map.on('pointermove', function(evt) {
  let el = document.getElementById('name');
  el.innerHTML = '';

  let coordinate = evt.coordinate;
  el.innerHTML += 'Coordinates: ' + ol.proj.toLonLat(coordinate) + '<br>';

  let pixel = map.getPixelFromCoordinate(coordinate);
  map.forEachFeatureAtPixel(pixel, function(feature) {
    if (feature.get('name') === undefined) {
      el.innerHTML += 'Capital: ' + feature.get('city') + '<br>';
    }
    else {
      el.innerHTML += feature.getProperties().name + '<br>';
    }
  });
});
```



# Interacción con capas vectoriales de datos

Hay diferentes maneras de programar interacción con el mapa. Una funcionalidad básica es mostrar la típica mano de enlaces en vez de la flecha del ratón cuando un usuario se mueve encima de un *feature* vectorial. Mira el funcionamiento (y el de los siguientes códigos) en el ejemplo [barcelona\\_vector\\_style.html](#):

```
map.on('pointermove', function(evt) {  
  map.getTargetElement().style.cursor = map.hasFeatureAtPixel(evt.pixel) ?  
  'pointer' : '';  
});
```



# Interacción con capas vectoriales de datos

La función *map.forEachFeatureAtPixel()* recoge los features existentes en el píxel clickado, la podemos usar para recoger información sobre un feature seleccionado:

```
map.on('click', function(evt) {  
  map.forEachFeatureAtPixel(evt.pixel, function(feature) {  
    console.log(feature.get('NOM'));  
  });  
});
```



# Interacción con capas vectoriales de datos

La forma correcta en la metodología de OpenLayers es usar las interacciones predefinidas. Solamente vamos a probar la más sencilla, que es Select, pero ya interacciones muy potentes para dibujar (*Draw*), arrastrar (*DragRotate*) y modificar features (*Modify*):

```
let selectMove = new ol.interaction.Select({
  condition: ol.events.condition.pointerMove,
  layers: [
    barrisLayer
  ],
  style: vectorStyleFunction
});

map.addInteraction(selectMove);

selectMove.on('select', function(evt) {
  if (evt.selected.length > 0)
    selectedFeature = evt.selected[0].get("NOM");
  else
    selectedFeature = null;
});
```



# Interacción con capas vectoriales de datos

Definimos un estilo básico sin texto, que mostramos por definición, y un estilo para los distritos seleccionados:

```
let vectorStyle = new ol.style.Style({
  stroke: new ol.style.Stroke({
    color: '#3399CC',
    width: 1.25
  }),
  fill: new ol.style.Fill({
    color: 'rgba(255,255,255,0.4)'
  }),
});

let vectorHighlightStyle = new ol.style.Style({
  stroke: new ol.style.Stroke({
    color: '#f00',
    width: 2,
  }),
  fill: new ol.style.Fill({
    color: 'rgba(255,255,255,0.4)',
  }),
  text: textStyle,
  zIndex: 1
});
```





# Interacción con capas vectoriales de datos

El estilo de texto define un color *stroke* de fondo para mostrar una sombra blanca. El color de texto se define con un *fill*, en este ejemplo no lo especificamos y por lo tanto se usa el color negro por definición:

```
let textStyle = new ol.style.Text({  
  font: '12px sans-serif',  
  stroke: new ol.style.Stroke({  
    color: '#fff',  
    width: 3  
  }),  
  overflow: true  
});
```



# Interacción con capas vectoriales de datos

Modificamos la función *vectorStyleFunction* para que comprueba si se trata de un feature seleccionado. En caso afirmativo añadimos el nombre del distrito al estilo antes de devolverlo, en caso negativo devolvemos el estilo por definición sin texto.

```
function vectorStyleFunction(feature, resolution) {  
  if (selectedFeature === feature.get('NOM')) {  
    textStyle.setText(feature.get('NOM'));  
    vectorHighlightStyle.setText(textStyle);  
    return vectorHighlightStyle;  
  }  
  else {  
    return vectorStyle;  
  }  
}
```



## Capas WMS

WMS (Web Map Service) es un estándar de la OCG (Open Geospatial Consortium). Habitualmente devuelven teselas de bitmaps. Aquí algunos servicios de entidades locales:

- ICGC:  
<https://www.icgc.cat/Administracio-i-empresa/Serveis/Geoinformacio-en-linia-Geoserveis>
- Diputación:  
<https://www.diba.cat/es/web/idebarcelona/serveis-de-mapes-wms>
- IDEE: <https://www.idee.es/web/guest/directorio-de-servicios>
- Catastro:  
<https://www.sedecatastro.gob.es/Accesos/SECACcDescargaDatos.aspx>



# Capas WMS

Probamos el topográfico del ICGC usando la siguiente definición de la capa:

```
new ol.layer.Tile({  
  source: new ol.source.TileWMS({  
    url: 'http://geoserveis.icc.cat/icc_mapesmultibase/utm/wms/service?',  
    params: {'LAYERS': 'topogris', 'VERSION': '1.1.1'}  
  })  
}),
```

Para conocer los detalles de un servicio WMS se utiliza la petición GetCapabilities. Suele tener este formato:

[http://geoserveis.icc.cat/icc\\_mapesmultibase/utm/wms/service?request=GetCapabilities](http://geoserveis.icc.cat/icc_mapesmultibase/utm/wms/service?request=GetCapabilities)



# Ejercicio

Carga el catastro con una capa teselada usando esta fuente:

```
source: new ol.source.TileWMS({  
  url: 'http://ovc.catastro.meh.es/Cartografia/WMS/ServidorWMS.aspx',  
  params: {  
    'LAYERS': 'catastro',  
    'TILED': true,  
    'SRS': 'EPSG:3857'  
  },  
})
```



# Capas WMS: GetFeatureInfo

La petición GetFeatureInfo de WMS permite identificar features y recibir información acerca de ellos desde el servidor. Funciona bien con capas del ICGC:

```
map.on('singleclick', function (evt) {
  document.getElementById('info').innerHTML = '';
  const viewResolution = /** @type {number} */ (map.getView().getResolution());

  const url = icgcSource.getFeatureInfoUrl(
    evt.coordinate,
    viewResolution,
    'EPSG:3857',
    {'INFO_FORMAT': 'text/html'}
  );

  if (url) {
    fetch(url)
      .then((response) => response.text())
      .then((html) => {
        document.getElementById('info').innerHTML = html;
      });
  }
});
```



# Capas WMS: GetFeatureInfo

Probamos GetFeatureInfo con el servidor WMS del catastro:

```
let catastroSource = new ol.source.TileWMS({  
  url: 'https://ovc.catastro.meh.es/Cartografia/WMS/ServidorWMS.aspx',  
  params: {  
    'LAYERS': 'catastro',  
    'TILED': true,  
    'SRS': 'EPSG:3857'  
  },  
})
```

Aquí están descritos los detalles del servicio WMS del catastro:

<http://www.catastro.meh.es/esp/wms.asp>

Actualmente las peticiones de *GetFeatureInfo* dan un fallo CORS y no son posibles, a pesar de añadir “no-cors” a las peticiones:

```
fetch(url, { mode: 'no-cors' })
```



# Capas WMS: GetFeatureInfo

Probamos GetFeatureInfo con el servidor WMS del ICGC:

```
let icgcSource = new ol.source.TileWMS({  
  url: 'https://geoserveis.icgc.cat/icc_bt5m/wms/service?',  
  params: {'LAYERS':  
    '02_ALTI_PA,03_POBL_PA,04_POBL_PA,05_VEGE_PA,06_VEGE_PA,07_POBL_PA,08_VEGE_PA,09_  
HIDR_PA,10_HIDR_PA,11_POBL_PA,12_POBL_PA,13_HIDR_PA,14_ALTI_LN,15_ALTI_LN,16_ALTI_  
_LN,17_ALTI_LN,19_HIDR_LN,20_HIDR_LN,21_HIDR_LN,22_HIDR_LN,23_VEGE_LN,24_VEGE_LN,  
25_ALTI_LN,26_POBL_LN,27_POBL_LN,28_POBL_LN,29_POBL_LN,30_POBL_PN,31_VIES_LN,32_V_  
IES_LN,33_VIES_LN,34_VIES_LN,35_VIES_LN,36_VIES_LN,37_VIES_LN,38_POBL_LN,39_POBL_  
_LN,40_TOPO_TX,41_PREF_PNTX,42_ALTI_PNTX,43_ALTI_TX', 'VERSION': '1.3.0'}  
})
```

Aquí están descritos los detalles del servicio WMS del topográfico del ICGC.

El ejemplo *barcelona\_wms\_getfeatureinfo.html* contiene el código necesario para mostrar la funcionalidad del *GetFeatureInfo*.





# Animaciones

La clase *View* viene con animaciones predefinidas, aquí un ejemplo que muestra algunas funcionalidades que probaremos en seguida.

Lanzamos la función animate() para hacer un zoom a la posición clickeada, aquí el código del ejemplo *rios\_animacion.html*:

```
map.on('singleclick', function(evt) {  
    map.getView().animate({  
        zoom: map.getView().getZoom()+1,  
        center: evt.coordinate  
    });  
});
```



# Animaciones

Para limitar la animación a los features clickeadas, buscamos los features bajo el click usando la función forEachFeatureAtPixel:

```
map.on('singleclick', function(evt) {  
    map.forEachFeatureAtPixel(evt.pixel, function(feature, layer) {  
        map.getView().animate({  
            zoom: map.getView().getZoom()+1,  
            center: evt.coordinate  
        });  
    });  
});
```



# Animaciones

El evento *postrender* se dispara cuando el mapa esta completamente renderizado. Lanzamos una animación cuando el mapa esta listo por primera vez.

```
let first = true;

map.on('postrender', function(){

  if (first) {

    map.getView().animate({
      center: ol.proj.fromLonLat([-1, 42]),
      zoom: 8,
      duration: 5000,
      easing: ol.easing.easeOut
    });

    first = false;
  }
});
```



# Animaciones

Aquí una función para detectar que el ratón esta encima de un feature y así podemos cambiar el cursor del ratón.

```
map.on('pointermove', function(evt) {  
    map.getTargetElement().style.cursor = map.hasFeatureAtPixel(  
        map.getEventPixel(evt.originalEvent),  
        {hitTolerance: 5}  
    ) ? 'pointer' : '';  
});
```

Barcelon**a**ctiva



Ajuntament  
de Barcelona

[bcn.cat/barcelonactiva](http://bcn.cat/barcelonactiva)  
[bcn.cat/cibernarium](http://bcn.cat/cibernarium)