

GIS Course

June 18-29 2018

Politecnico di Milano, Lecco Campus



POLITECNICO
MILANO 1863



Geodesy and geoinformatics for
sustainable development in Jordan
586070-EPP-1-2017-1-SE-EPPKA2-CBHE-JP

OpenLayers

Candan Eylül Kilsedar

28/06/2018

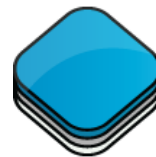
Politecnico di Milano, Department of Civil and Environmental Engineering,
Piazza Leonardo da Vinci 32, 20133 Milan, Italy



<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

OpenLayers Introduction

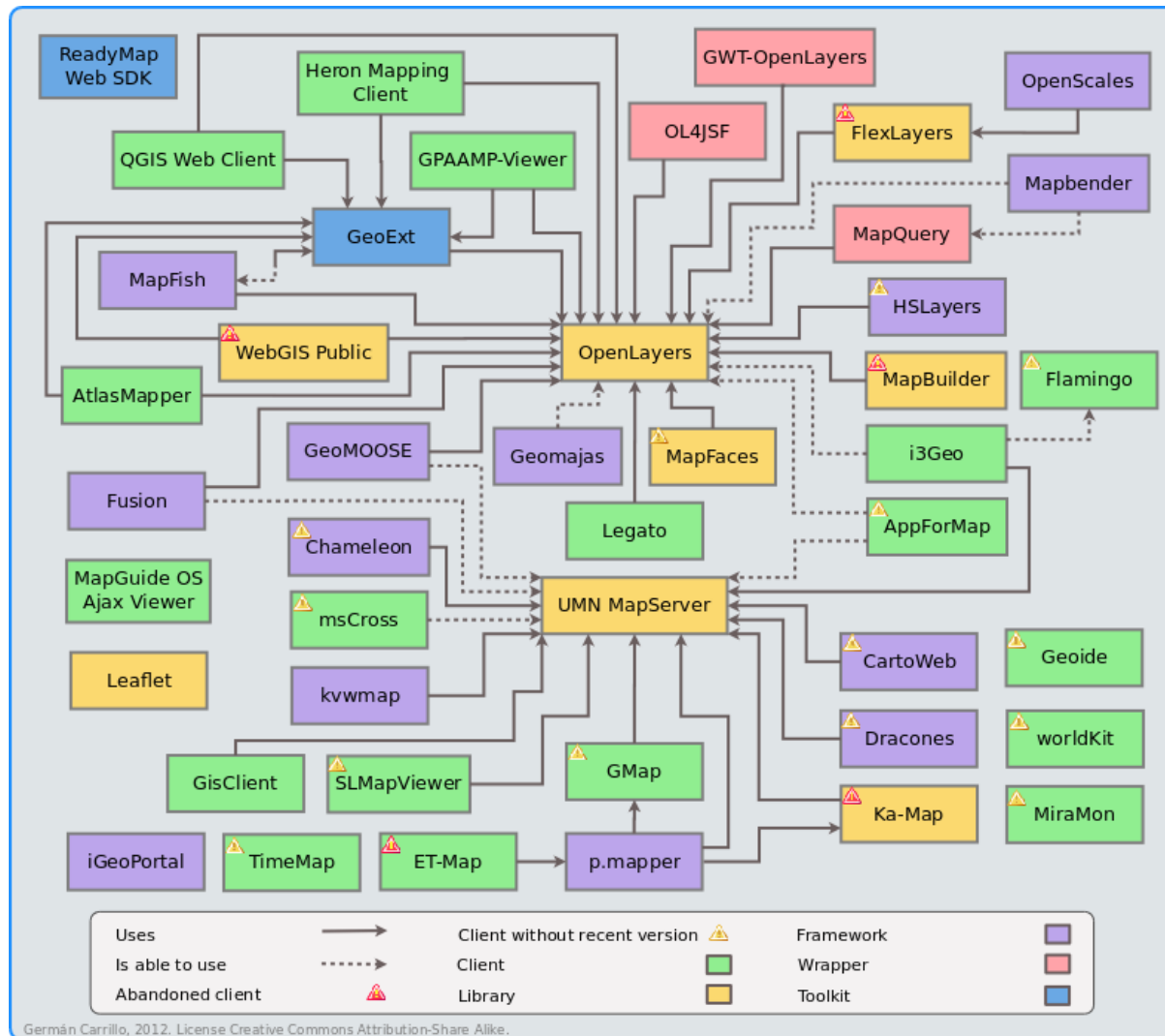
- ✓ Created by the US company MetaCarta in 2005, OpenLayers is a **JavaScript-based mapping library** allowing users to display dynamic maps in web browsers without any server-side dependency.
- ✓ It serves as the foundation of most of the common web mapping interfaces. OpenLayers is completely **free and open source software (FOSS)** and it is provided under the [2-clause BSD License](#) (also known as the FreeBSD).
- ✓ It is an official project of the Open Source Geospatial Foundation (**OSGeo**) since 2007; it is developed and supported by many organizations around the world.
- ✓ OpenLayers implements a JavaScript API to create rich web-based geographic applications similar to the Google Maps and Bing Maps APIs, but with an important difference: it is free and open source software!



OpenLayers



Overview on FOSS Web Mapping Clients



source: Carrillo 2012 - <http://geotux.tuxfamily.org/index.php/en/geo-blogs/item/291-comparacion-clientes-web-v6>



Other Popular Web Mapping Clients



Mapbox




OpenLayers Introduction

- ✓ Most important capabilities are:
 - ✓ overlap of many standards-compliant map layers in a single application;
 - ✓ displaying **images/tiles** from WMS, WMTS, Bing Maps, OpenStreetMap, Mapbox, Stamen and any other XYZ source;
 - ✓ **vector** data rendering and styling with support for KML, GML, GeoJSON, TopoJSON, Mapbox vector tiles and a growing number of other formats;
 - ✓ support for **feature editing** (insert, update and delete) and writing back the changes to the server through a Web Feature Service-Transaction (WFS-T);
 - ✓ can be used with any JavaScript library (jQuery, Ext JS, ...);
 - ✓ targets **HTML5** and **CSS3** (so that map is rendered using WebGL and canvas, styling map controls with CSS is possible);
 - ✓ supports **mobile**;
 - ✓ supports integration with the [Cesium](#) library to enable full virtual globe capabilities;
 - ✓ full documentation (<http://openlayers.org/en/latest/apidoc/>).



OpenLayers Examples

- ✓ Best documentation is the example gallery (<http://openlayers.org/en/latest/examples/>).

 **OpenLayers Examples** (166) Docs Examples API Code

Accessible Map
(accessible.html)
Example of an accessible map.

View Animation
(animation.html)
Demonstrates animated pan, zoom, and rotation.

Image ArcGIS MapServer
(arcgis-image.html)
Example of an image ArcGIS layer.

Tiled ArcGIS MapServer
(arcgis-tiled.html)
Example of a tiled ArcGIS layer.

Attributions
(attributions.html)
Example of a attributions visibly change on map resize, to collapse them on small maps.

Bing Maps
(bing-maps.html)
Example of a Bing Maps layer.

Blend Modes
(blend-modes.html)
Shows how to change the canvas compositing / blending mode in post- and precompose eventhandlers.

Box Selection
(box-selection.html)
Using a DragBox interaction to select features.

Custom Tooltips
(button-title.html)
This example shows how to customize the buttons tooltips with Bootstrap.

Styling feature with CanvasGradient or CanvasPattern
(canvas-gradient-pattern.html)
Example showing the countries vector layer styled with patterns and gradients.

Canvas Tiles
(canvas-tiles.html)
Renders tiles with coordinates for debugging.

CartoDB source example
(cartodb.html)
Example of a cartodb map.



Web Development

- ✓ Mainly three core technologies are used to develop web pages:
 - ✓ **HTML** (HyperText Markup Language) is a markup language to structure and present content.
 - ✓ **CSS** (Cascading Style Sheets) is a style sheet language used to style the content.
 - ✓ **JavaScript** is a high-level programming language that is used to define the behavioral aspects.



Web Development

- ✓ A web server processes requests via **HTTP** (Hypertext Transfer Protocol). Primarily it stores and delivers web pages to clients.

A thorough explanation is available at https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server.

- ✓ **Apache2** web server is already installed in OSGeo-Live.
- ✓ Apache2 looks for the files that make up the website in the directory **/var/www/html**.
- ✓ To see the permissions of this directory run in terminal:

```
ls -l /var/www
```

- ✓ Notice that it is **rwxr-xr-x**, which means only **root** can write (**w**) inside the html directory. For this reason you need to use terminal to work using **sudo**, or use **nautilus**. Install and use nautilus in the following way:

```
sudo apt install nautilus  
sudo nautilus /var/www/html
```



Web Development

- ✓ Run the commands below or use nautilus.

```
cd /var/www/html
sudo mkdir geo4d-web
cd geo4d-web
sudo touch index.html
sudo apt install gedit
sudo gedit index.html
```

- ✓ **Leafpad** is the text editor available in OSGeo-Live.
- ✓ **gedit** supports color highlighting, which eases understanding the code.
- ✓ Alternatively, you can use [Sublime Text](#) for a modern and sophisticated text editor. Then use `sudo subl /var/www/html/geo4d-web` to open the project folder with correct permissions.
- ✓ Check <http://localhost/geo4d-web> to see the web page.
- ✓ You can use [Codecademy](#) or similar platforms for further learning web development, [Learn JavaScript](#) for further learning JavaScript.



Inspection Tools

- ✓ Just right-click on the page, and then select “Inspect Element”.
- ✓ Inspectors allow to:
 - ✓ check the console for errors, warnings, ...;
 - ✓ inspect HTML and modify style and layout in real time;
 - ✓ use the JavaScript debugger;
 - ✓ analyze network usage and performance;
 - ✓ check the Document Object Model (DOM).



Web Development - HTML

- ✓ HTML defines the structure and layout of a web page and how a web page looks using **tags** that have **attributes**.
- ✓ Example structures could be head, body, images, tables, lists, paragraphs, links, blocks and so on.
- ✓ HTML tags are keywords (tag names) surrounded by angle brackets:
`<tag name>content</tag name>`
- ✓ HTML tags most of the time come in pairs. First tag is called the **opening tag** and second tag is called the **closing tag**. Closing tag is same as the opening tag, but with a slash before the tag name.
- ✓ For example `<p>` means a paragraph, and a paragraph most basically can be defined as:
`<p>This a paragraph.</p>.`



Web Development - HTML

- ✓ A basic HTML element can be structured as follows:

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
</head>
<body>
  <p>This is my first paragraph.</p>
</body>
</html>
```

- ✓ The doctype declaration defines the document type to be HTML.
- ✓ Inside the meta tag the character set (character encoding) is defined, which is required to display an HTML page correctly.
- ✓ The text between `<html>` and `</html>` describes an HTML document.
- ✓ The text between `<head>` and `</head>` provides information about the document.
- ✓ The text between `<body>` and `</body>` describes the visible page content.
- ✓ The text between `<title>` and `</title>` provides a title for the document.
- ✓ The text between `<p>` and `</p>` describes a paragraph.



Web Development - CSS

- ✓ CSS describes how HTML elements are to be displayed on screen.
- ✓ There are three ways of inserting a style sheet:
 - ✓ external style sheet

index.html

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <link rel="stylesheet" href="main.css">
</head>
<body>
  <p>This is my first paragraph.</p>
</body>
</html>
```

main.css

```
p {
  color: blue;
}
```



Web Development - CSS

✓ internal style sheet

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    p {
      color: blue;
    }
  </style>
</head>
<body>
  <p>This is my first paragraph.</p> </body>
</html>
```

✓ inline style

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
</head>
<body>
  <p style="color: blue;">This is my first paragraph.</p>
</body>
</html>
```



Web Development - CSS

- ✓ CSS **selectors** are patterns to select element(s) to style. There are three main ways to select:

- ✓ **element** selector

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    p {
      color: blue;
    }
  </style>
</head>
<body>
  <p>This is my first paragraph.</p>
</body>
</html>
```



Web Development - CSS

✓ id selector

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    #p-blue {
      color: blue;
    }
  </style>
</head>
<body>
  <p id="p-blue">This is my first paragraph.</p>
</body>
</html>
```

✓ class selector

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    .p-blue {
      color: blue;
    }
  </style>
</head>
<body>
  <p class="p-blue">This is my first paragraph.</p>
</body>
</html>
```



Web Development - JavaScript

- ✓ Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern web browsers without plug-ins.

- ✓ JavaScript is also used in environments that are not web-based.

- ✓ In HTML, JavaScript code must be inserted between `<script>` and `</script>` tags in `<head>` or `<body>`.

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    #p-blue {
      color: blue;
    }
  </style>
</head>
<body>
  <p id="p-blue">This is my first paragraph.</p>
  <script>
    document.getElementById("p-blue").addEventListener("click", myFunction);

    function myFunction() {
      alert("Paragraph clicked!");
    }
  </script>
</body>
</html>
```



Web Development - JavaScript

- ✓ Moreover, it can be added **externally**.

index.html

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>Page Title</title>
  <style>
    #p-blue {
      color: blue;
    }
  </style>
</head>
<body>
  <p id="p-blue">This is my first paragraph.</p>
  <script src="main.js"></script>
</body>
</html>
```

main.js

```
document.getElementById("p-blue").addEventListener("click", myFunction);

function myFunction() {
  alert("Paragraph clicked!");
}
```



Installation

- ✓ Using a web server (such as Apache2) it is possible to get layers from a local or remote map server (e.g. GeoServer, MapServer, ...).
- ✓ We will be working under `/var/www/html` folder. Create a folder called `geo4d-openlayers`.
- ✓ Download the latest version (v4.6.5.zip) of OpenLayers, available at <https://openlayers.org/download/>.
- ✓ It is a good approach to download the libraries and place them under the web server.
- ✓ Under `geo4d-openlayers`, create a new folder for libraries called `libraries` and place the unzipped OpenLayers library in it.
- ✓ Moreover, create two files called `index.html` and `map.js` to start building the map client under `geo4d-openlayers`.



Getting Started

index.html

```
<!doctype html>
<html lang="en">
<meta charset="utf-8">
<head>
  <title>WRITE HERE THE TITLE OF THE PAGE</title>
  <link rel="stylesheet" href="libraries/ol_v4.6.5/css/ol.css">
  <script src="libraries/ol_v4.6.5/build/ol.js"></script>
</head>
<body>
  <h2><b><center>WRITE HERE THE HEADING OF THE PAGE</center></b></h2>
  <div id="map"></div>
  <script src="map.js"></script>
</body>
</html>
```

map.js

```
var map = new ol.Map({
  target: document.getElementById('map')
});
```



Getting Started

- ✓ **ol.Map** is the core component of OpenLayers. For a map to render, a view, one or more layers, and a target container are needed.
- ✓ **target** defines the container for the map, either the element itself or the id of the element. Here the id of the created div element is used.

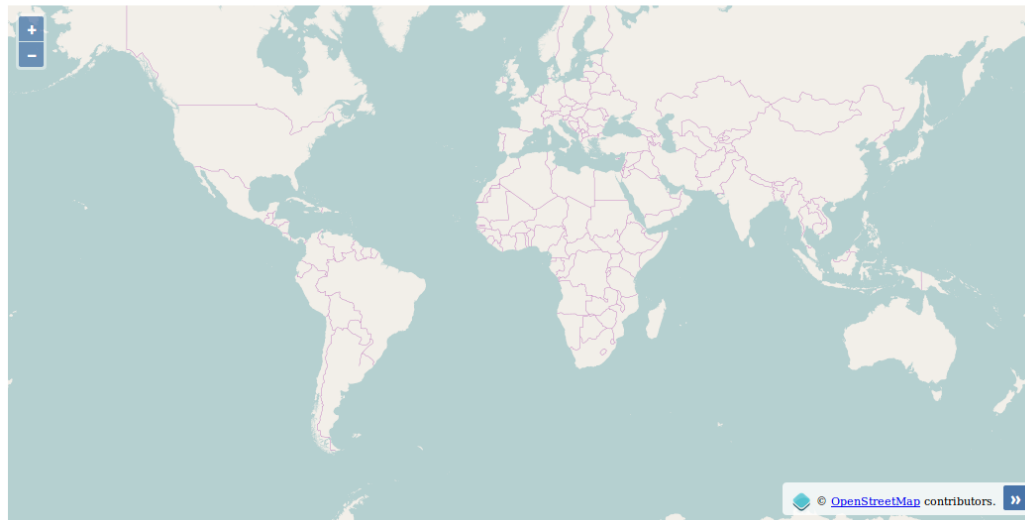
```
var map = new ol.Map({  
  target: document.getElementById('map'),  
  layers: [  
    new ol.layer.Tile({  
      visible: true,  
      source: new ol.source.OSM()  
    })  
  ],  
  view: new ol.View({  
    center: [0, 0],  
    zoom: 2  
  })  
});
```



Getting Started

- ✓ The **layers** defines all the layers added to the map. In our example, we added OpenStreetMap (OSM) which is a tile layer using **ol.layer.Tile** object. This object defines layer sources that provide pre-rendered, tiled images in grids that are organized by zoom levels for specific resolutions.
- ✓ **ol.View** object represents a simple 2D view of the map. This is the object to act upon to change the center, resolution, and rotation of the map. Here the center is set to (0, 0) point and zoom is set to a level so that the whole world is visible.

OpenLayers Map



Adding a WMS Layer

- ✓ It is possible to add a layer published in GeoServer using WMS. Let's start with Ecuador boundary. The layer is **Ecuador:ECU_adm0**, which is the combination of **workspace** (comes first) and **layer name** (comes second), however it might be different in your case, so check your GeoServer!

```
var osm = new ol.layer.Tile({
  visible: true,
  source: new ol.source.OSM()
});
var ecuadorBoundary= new ol.layer.Image({
  source: new ol.source.ImageWMS({
    url: 'http://localhost:8082/geoserver/wms',
    params: {'LAYERS': 'Ecuador:ECU_adm0'}
  })
});
var map = new ol.Map({
  target: document.getElementById('map'),
  layers: [osm, ecuadorBoundary],
  view: new ol.View({
    center: [0, 0],
    zoom: 2
  })
});
```



Adding a WMS Layer

- ✓ **ol.Layer.Image** is an object for server-rendered images that are available for arbitrary extents and resolutions.
- ✓ **ol.source.ImageWMS** is the object for defining the source for WMS servers providing single, untiled images.
- ✓ **url** is the URL of the WMS service, which is the GeoServer WMS service used in the previous practice (<http://localhost:8082/geoserver/wms>).
- ✓ **params** is the WMS request parameters. At least a **LAYERS** parameter is required. **LAYERS** parameter defines the layers by their workspace (Ecuador) and their layer title (ECU_adm0).

OpenLayers Map



Adding a WMS Layer

- ✓ Change the center of the map from the world to Ecuador by changing the view of the map as follows:

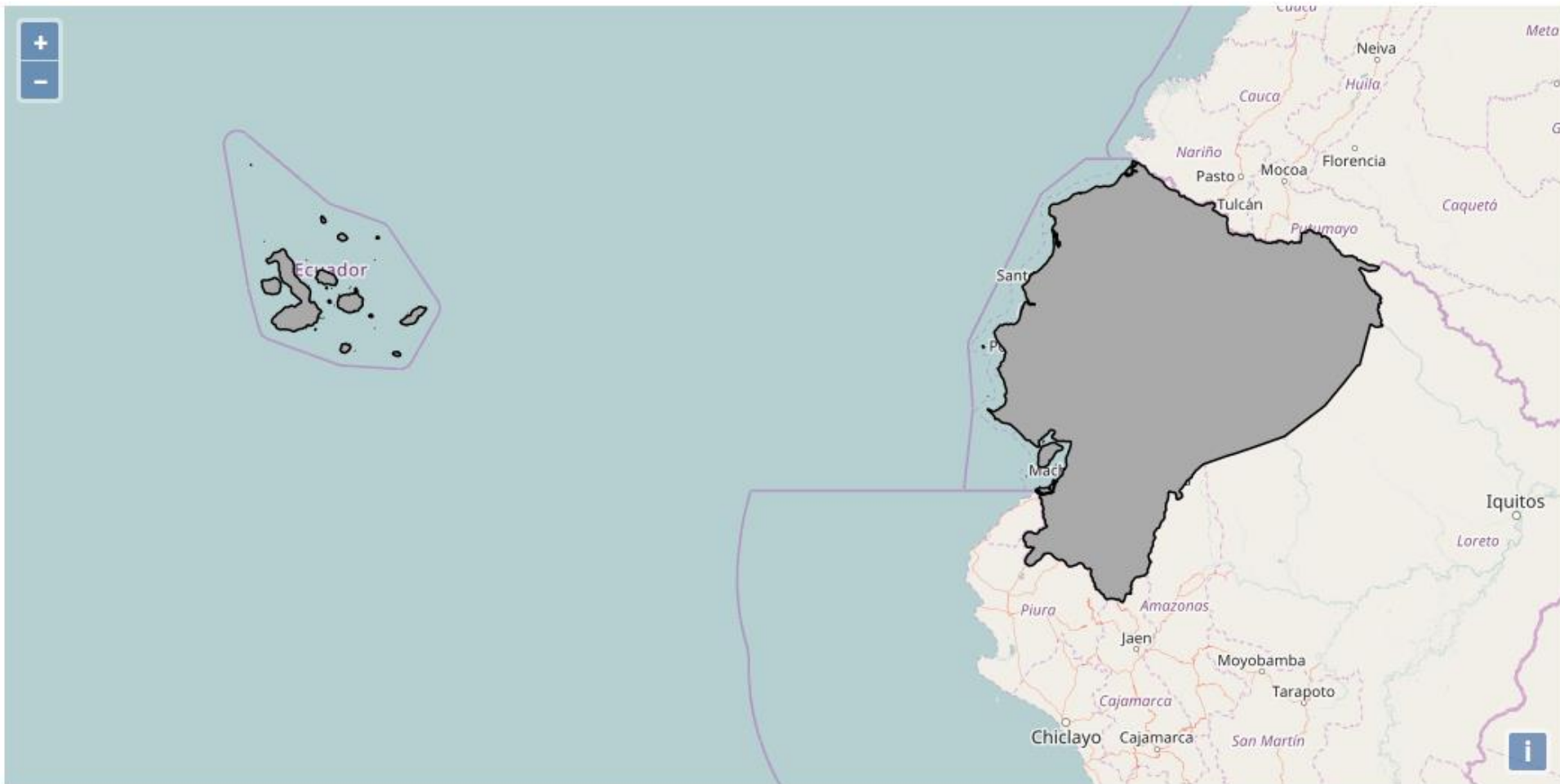
```
var map = new ol.Map({  
  target: document.getElementById('map'),  
  layers: [osm, ecuadorBoundary],  
  view: new ol.View({  
    center: ol.proj.fromLonLat([-84, -2]),  
    zoom: 6  
  })  
});
```

- ✓ The center of map viewer was previously set to (0, 0) point. We changed the coordinates to the coordinates of Ecuador in EPSG:4326 reference system which basically is latitude and longitude. However this needs to be transformed to EPSG:3857 which is the reference system of base map (OSM), which is why it is necessary to use `ol.proj.fromLonLat` method. The zoom level is set to a proper level.



Adding a WMS Layer

OpenLayers Map



Adding a WMS Layer

- ✓ It is also possible to add layers using **addLayer** method as follows:

```
var map = new ol.Map({  
  target: document.getElementById('map'),  
  view: new ol.View({  
    center: ol.proj.fromLonLat([-84, -2]),  
    zoom: 6  
  })  
});  
map.addLayer(osm);  
map.addLayer(ecuadorBoundary);
```

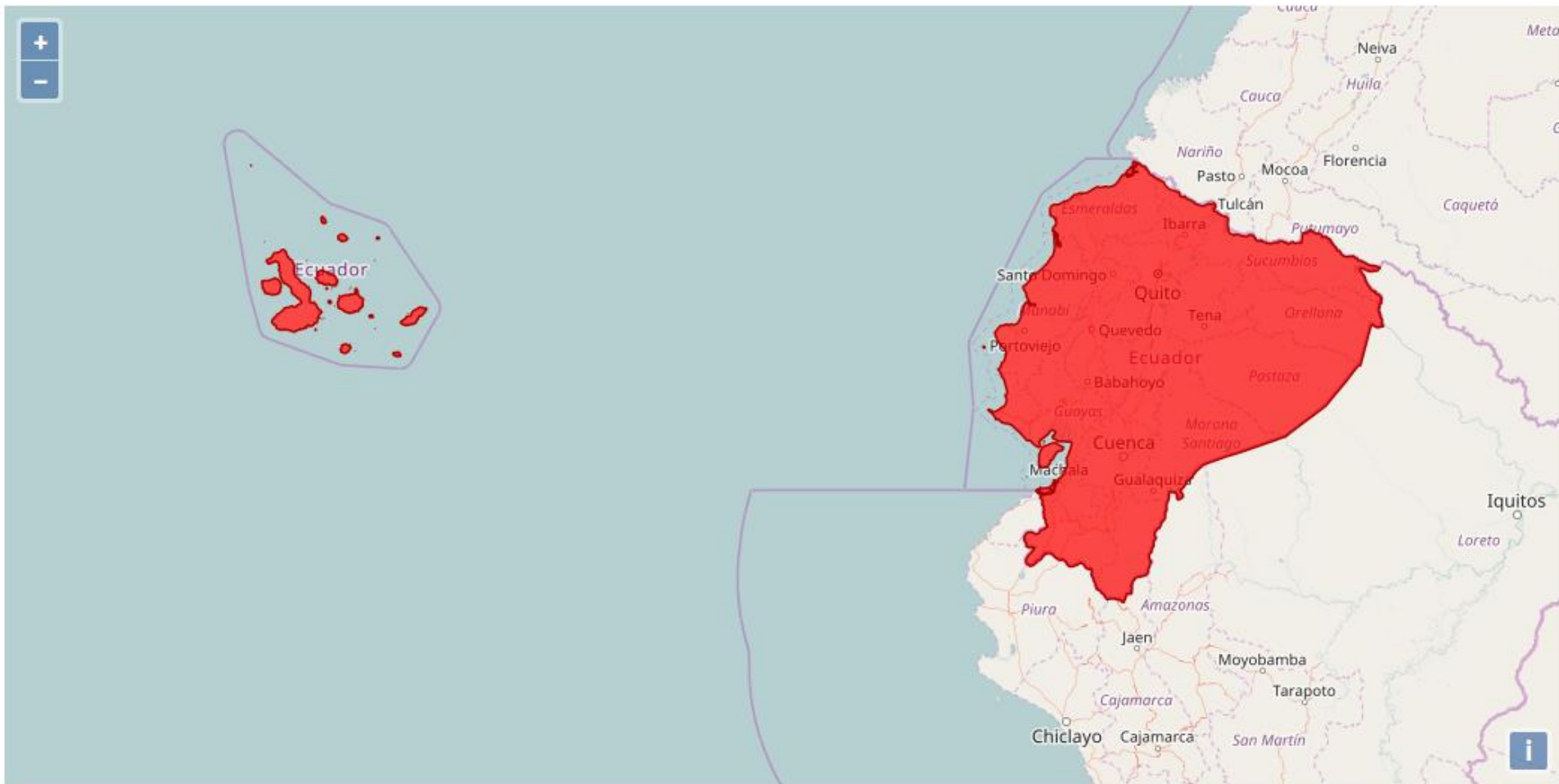
- ✓ The **Ecuador:ECU_adm0** layer is published by GeoServer with other available styles, e.g. the style **restricted** (check on GeoServer if this style is among the available ones for that layer). To use this style modify the layer definition as follows:

```
var ecuadorBoundary = new ol.layer.Image({  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:ECU_adm0', 'STYLES': 'restricted'}  
  })  
});
```



Adding a WMS Layer

OpenLayers Map



Adding WMS Layers

- ✓ Apart from Ecuador boundary (**ECU_adm0**), let's add also the layers published on GeoServer Ecuador provinces (**ECU_adm1**), Ecuador roads (**ECU_roads**), and Ecuador rivers (**ECU_water_lines**).

```
var ecuadorProvinces = new ol.layer.Image({  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:ECU_adm1'}  
  }),  
  opacity: 0.5  
});
```

```
var ecuadorRoads = new ol.layer.Image({  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:ECU_roads'}  
  }),  
  visible: false  
});
```



Adding WMS Layers

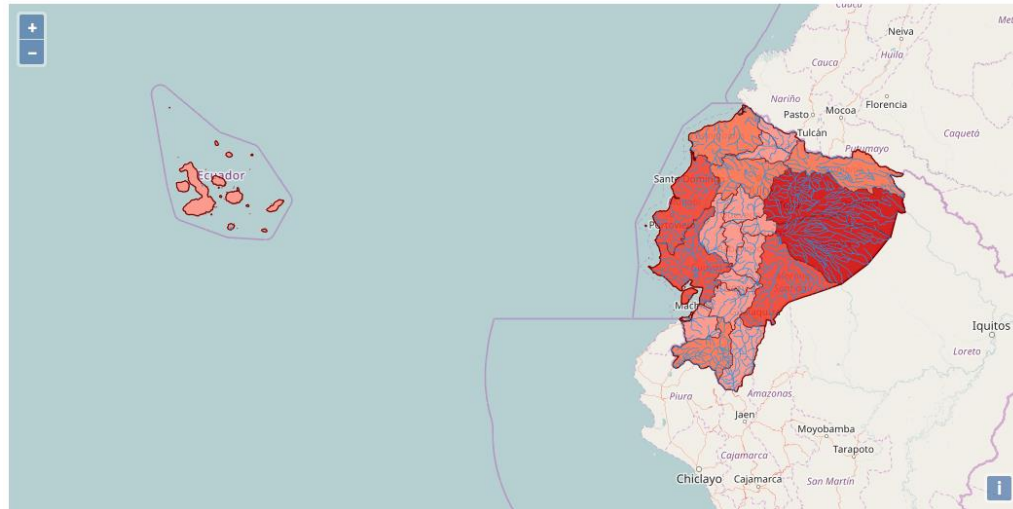
```
var ecuadorRivers = new ol.layer.Image({
  source: new ol.source.ImageWMS({
    url: 'http://localhost:8082/geoserver/wms',
    params: {'LAYERS': 'Ecuador:ECU_water_lines'}
  }),
  minResolution: 1000,
  maxResolution: 5000
});

var map = new ol.Map({
  target: document.getElementById('map'),
  layers: [osm, ecuadorBoundary, ecuadorProvinces, ecuadorRoads, ecuadorRivers],
  view: new ol.View({
    center: ol.proj.fromLonLat([-84, -2]),
    zoom: 6
  })
});
```



Adding WMS Layers

- ✓ The **opacity** value ranges between 0 and 1. The default value is 1.
- ✓ Every layer is visible by default. Setting **visible** parameter to false makes the layer invisible.
- ✓ The **minResolution** is the minimum resolution (inclusive) at which this layer will be visible and the **maxResolution** is the maximum resolution (exclusive) below which this layer will be visible. It is sometimes recommended to make some layers visible at a certain resolution level. So, in our case the rivers will be visible between resolution levels 1000 and 5000.



Adding Controls to the Map

- ✓ It is useful to add some controls to the map. To add control for the 2D coordinates of **mouse cursor** (by default in the top right corner), the indication of the current **map scale** (by default in the bottom left corner), **overview map** (by default in the bottom left corner) and the **full screen** control (by default in the top right corner) change the map definition as follows:

```
var map = new ol.Map({  
  target: document.getElementById('map'),  
  layers: [...],  
  view: new ol.View({...}),  
  controls: ol.control.defaults().extend([  
    new ol.control.ScaleLine(),  
    new ol.control.FullScreen(),  
    new ol.control.OverviewMap(),  
    new ol.control.MousePosition({  
      coordinateFormat: ol.coordinate.createStringXY(4),  
      projection: 'EPSG:4326'  
    })  
  ])  
});
```



Adding Controls to the Map

- ✓ To change the placement and look of mouse position and overview map controls, the style of them can be changed as follows (as they are overlapping with the other controls and the style of mouse position is different from the others):

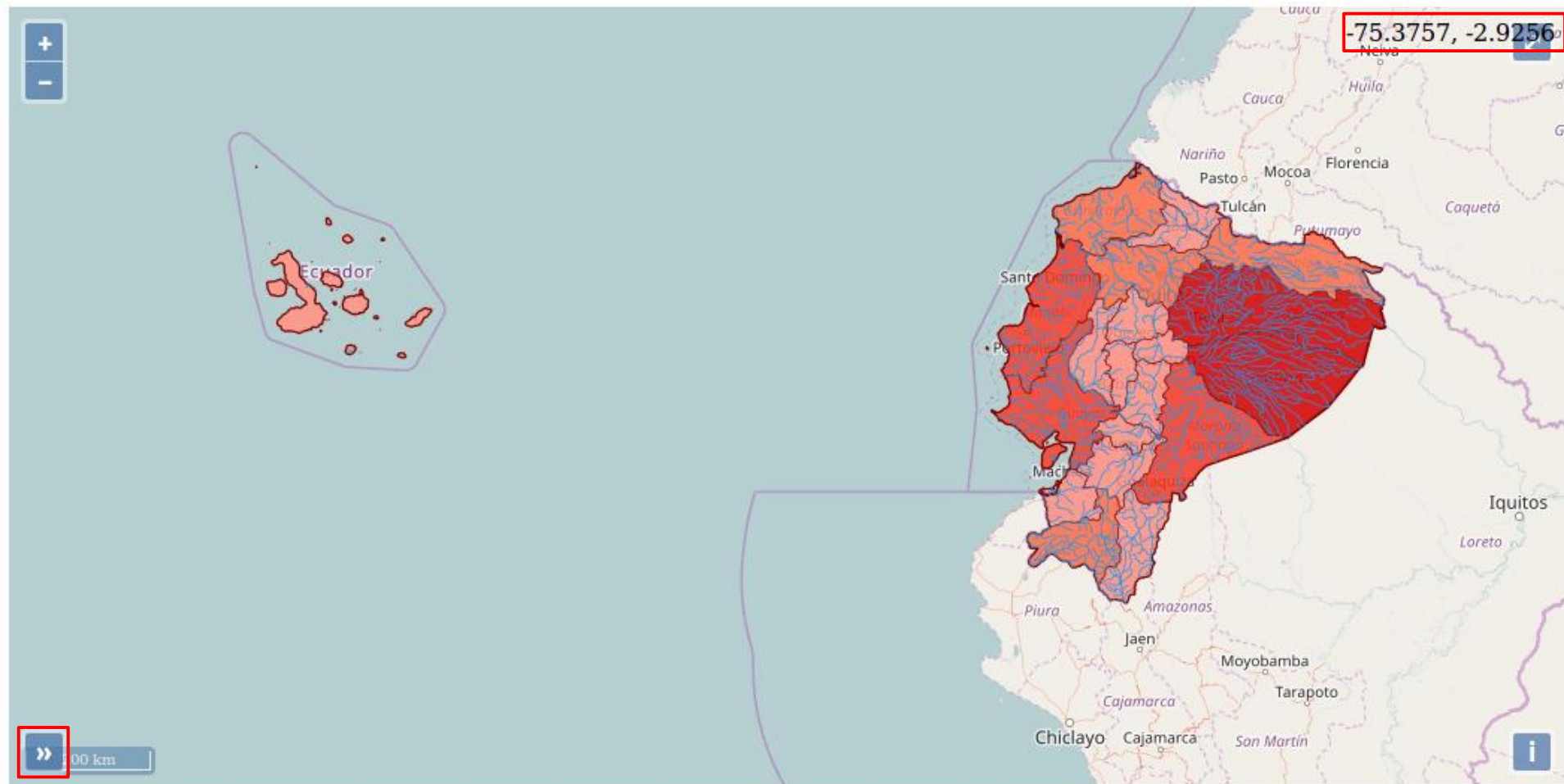
```
.ol-mouse-position {  
    right: 42px;  
    color: rgb(255, 255, 255);  
    font-size: 12px;  
    text-align: center;  
    padding: 2.5px;  
    background: rgba(0, 60, 136, 0.5);  
    background-clip: padding-box;  
    border: 3px solid rgba(255, 255, 255, 0.4);  
    border-radius: 4px;  
}  
.ol-overviewmap {  
    left: 8px;  
    bottom: 34px;  
}
```

- ✓ This piece of CSS code should be put inside a CSS file (`main.css`) and imported in the HTML file.
- ✓ Detailed information can be found at <http://www.w3schools.com/css/default.asp>.



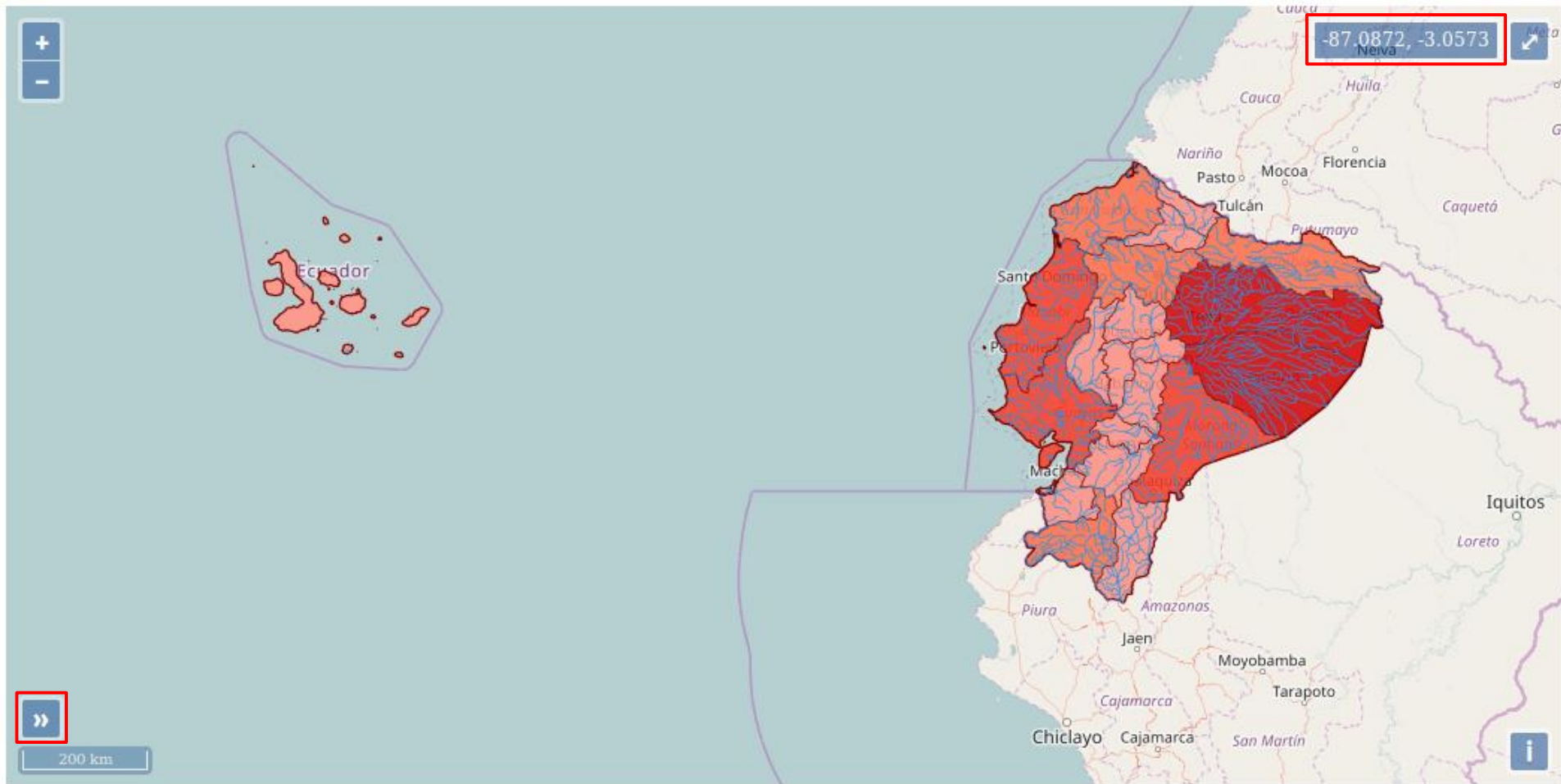
Adding Controls to the Map

- ✓ The map **before** changing the style of the controls:



Adding Controls to the Map

- ✓ The map **after** changing the style of the controls:



Adding Controls to the Map

- ✓ It is useful to add also a legend. Legend is not included in the OpenLayers library, so we use an external one, [olayerswitcher](http://olayerswitcher.com/). Import the library as follows:

```
<link rel="stylesheet" href="http://rawgit.com/walkermatt/ol-layerswitcher/master/src/ol-layerswitcher.css">
<script src="http://rawgit.com/walkermatt/ol-layerswitcher/master/dist/ol-layerswitcher.js"></script>
```

- ✓ Your **layers** should be changed to the following:

```
layers: [
  new ol.layer.Group({
    title: 'Basemaps',
    layers: [osm]
  }),
  new ol.layer.Group({
    title: 'Overlay Layers',
    layers: [ecuadorBoundary, ecuadorProvinces, ecuadorRoads,
             ecuadorRivers]
  })
]
```



Adding Controls to the Map

- ✓ Also add these lines after the definition of map:

```
var layerSwitcher = new ol.control.LayerSwitcher({});  
map.addControl(layerSwitcher);
```

- ✓ **ol.Layer.Group** is a collection of layers that are handled together.
- ✓ **title** is the title of the layer group.
- ✓ **layers** is an array of layers.
- ✓ All the layers that are intended to be listed in the legend must have a **title** attribute, and only the basemaps should have a **type** attribute. So the OSM basemap definition should be changed to:

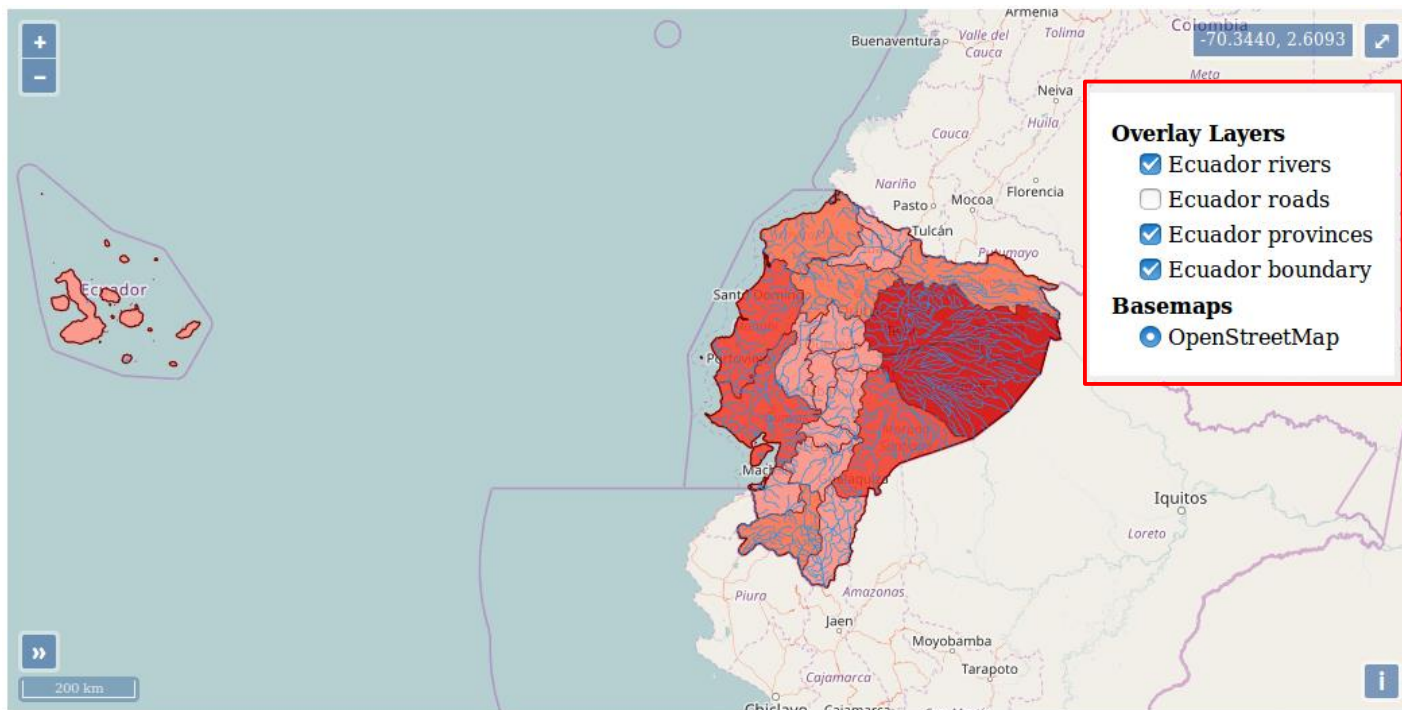
```
var osm = new ol.layer.Tile({  
  title: 'OpenStreetMap',  
  type: 'base',  
  visible: true,  
  source: new ol.source.OSM()  
});
```



Adding Controls to the Map

- ✓ An example of how the definition of an overlay layer should be changed:

```
var ecuadorBoundary = new ol.layer.Image({  
  title: 'Ecuador boundary',  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:ECU_adm0', 'STYLES': 'restricted'}  
  })  
});
```



Adding Bing Maps

- ✓ Let's now add Bing Maps (roads, aerial & aerial with labels) to the map. For this, write the following lines of code after the definition of OpenStreetMap:

```
var bingRoads = new ol.layer.Tile({  
  title: 'Bing Maps - Roads',  
  type: 'base',  
  visible: false,  
  source: new ol.source.BingMaps({  
    key: 'X',  
    imagerySet: 'Road'  
  })  
});
```

```
var bingAerial = new ol.layer.Tile({  
  title: 'Bing Maps - Aerial',  
  type: 'base',  
  visible: false,  
  source: new ol.source.BingMaps({  
    key: 'X',  
    imagerySet: 'Aerial'  
  })  
});
```



Adding Bing Maps

```
var bingAerialWithLabels = new ol.layer.Tile({  
  title: 'Bing Maps - Aerial with Labels',  
  type: 'base',  
  visible: false,  
  source: new ol.source.BingMaps({  
    key: 'X',  
    imagerySet: 'AerialWithLabels'  
  })  
});
```

- ✓ It is required to have a Bing Maps key to use the Bing Maps.
- ✓ Go to the link <https://msdn.microsoft.com/en-us/library/ff428642.aspx> and follow the steps under “Creating a Bing Maps Key”.
- ✓ After the key is created, instead of X in the definition of the source of the Bing layers use the key.



Adding Stamen

- ✓ It is also possible to add Stamen basemaps. We will add **watercolor** and **toner** styled OSM data. More can be found at <http://maps.stamen.com>.
- ✓ Add these lines of code after the definition of Bing Maps:

```
var stamenWatercolor = new ol.layer.Tile({  
  title: 'Stamen Watercolor',  
  type: 'base',  
  visible: false,  
  source: new ol.source.Stamen({  
    layer: 'watercolor'  
  })  
});
```

```
var stamenToner = new ol.layer.Tile({  
  title: 'Stamen Toner',  
  type: 'base',  
  visible: false,  
  source: new ol.source.Stamen({  
    layer: 'toner'  
  })  
});
```



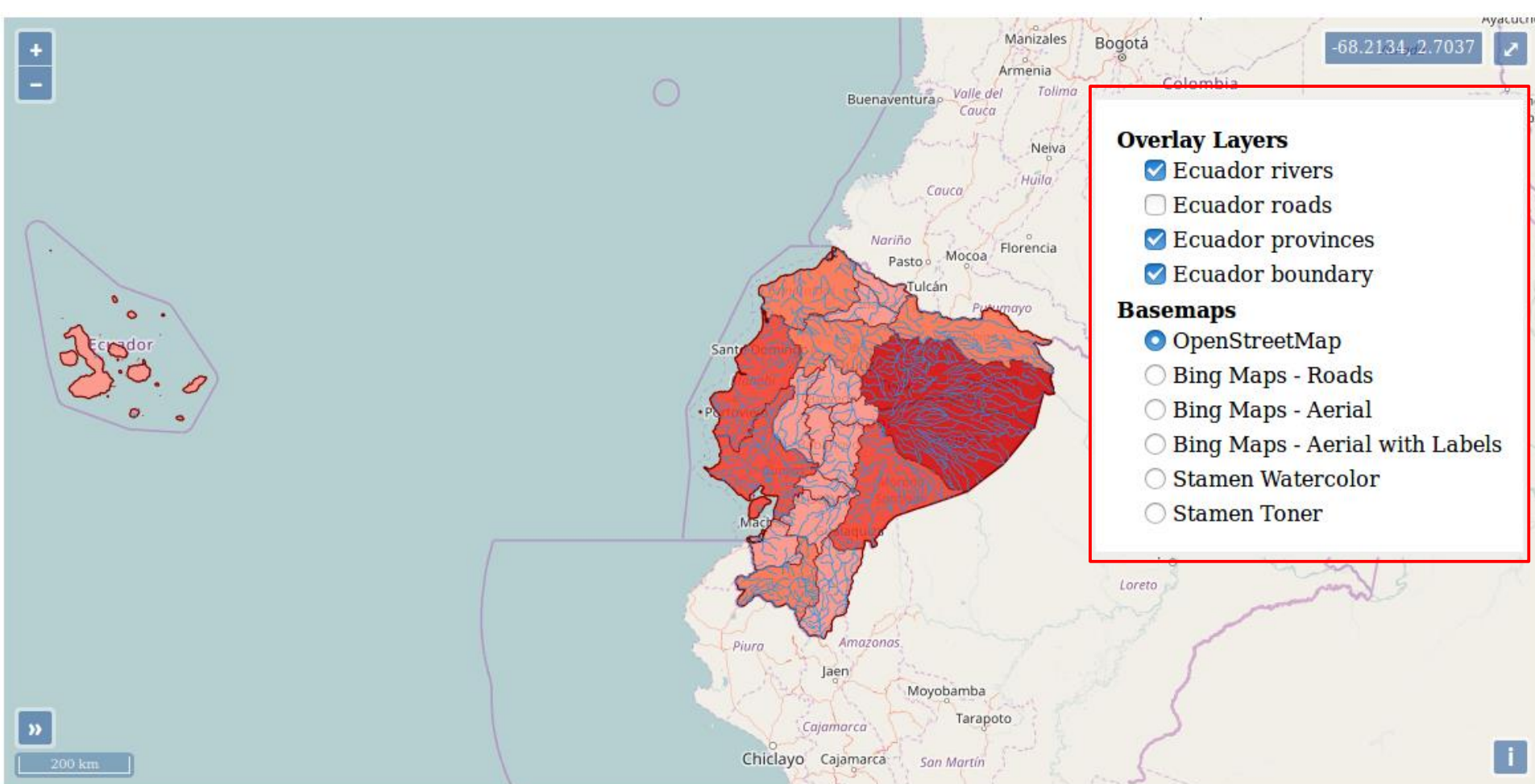
Adding Controls to the Map

- ✓ Do not forget to change the layers array inside the map definition as follows and add the newly defined layers :

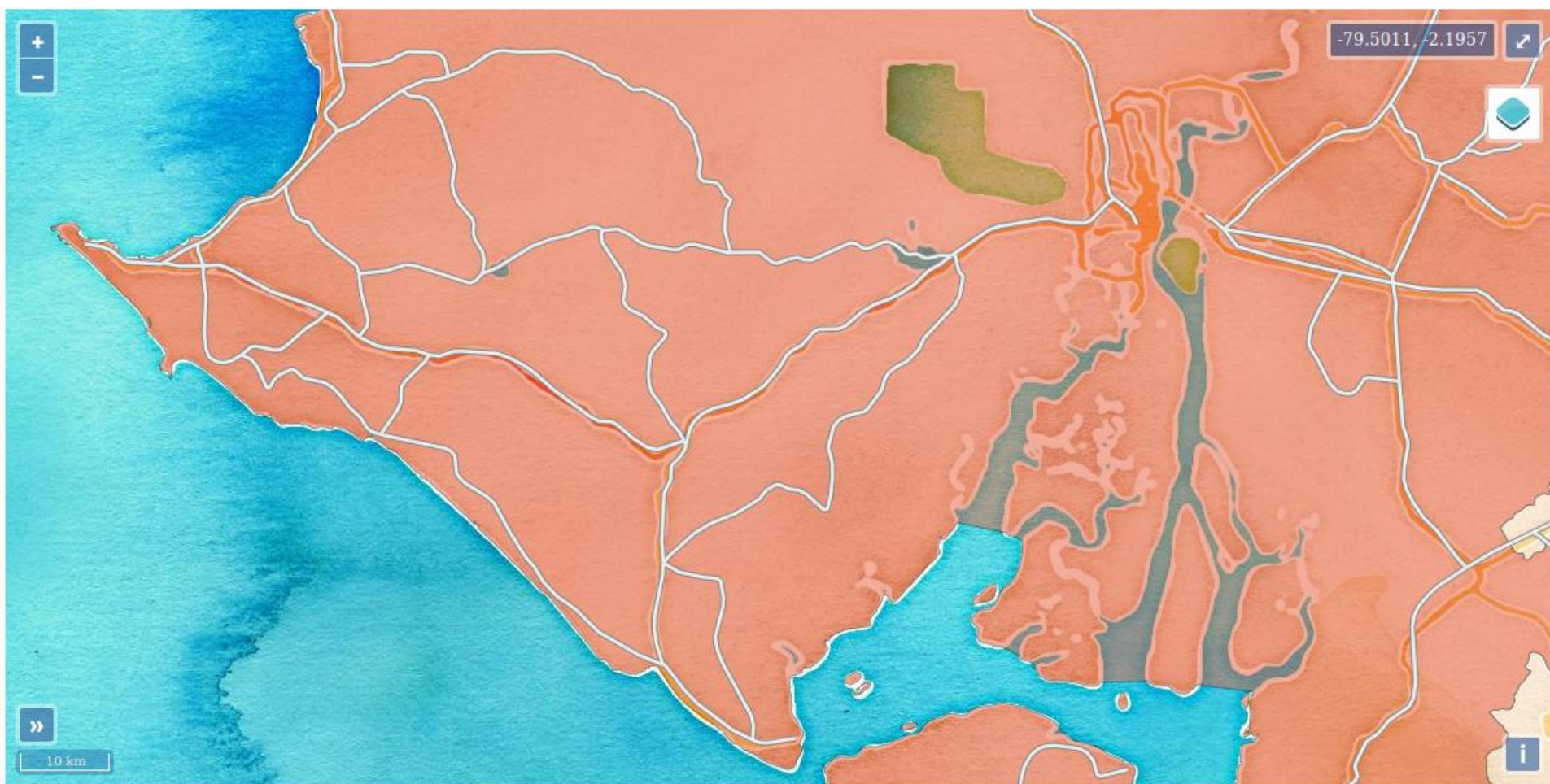
```
layers: [  
  new ol.layer.Group({  
    title: 'Basemaps',  
    layers: [stamenToner, stamenWatercolor, bingAerialWithLabels,  
bingAerial, bingRoads, osm]  
  }),  
  new ol.layer.Group({  
    title: 'Overlay Layers',  
    layers: [ecuadorBoundary, ecuadorProvinces, ecuadorRoads, ecuadorRivers]  
  })  
]
```



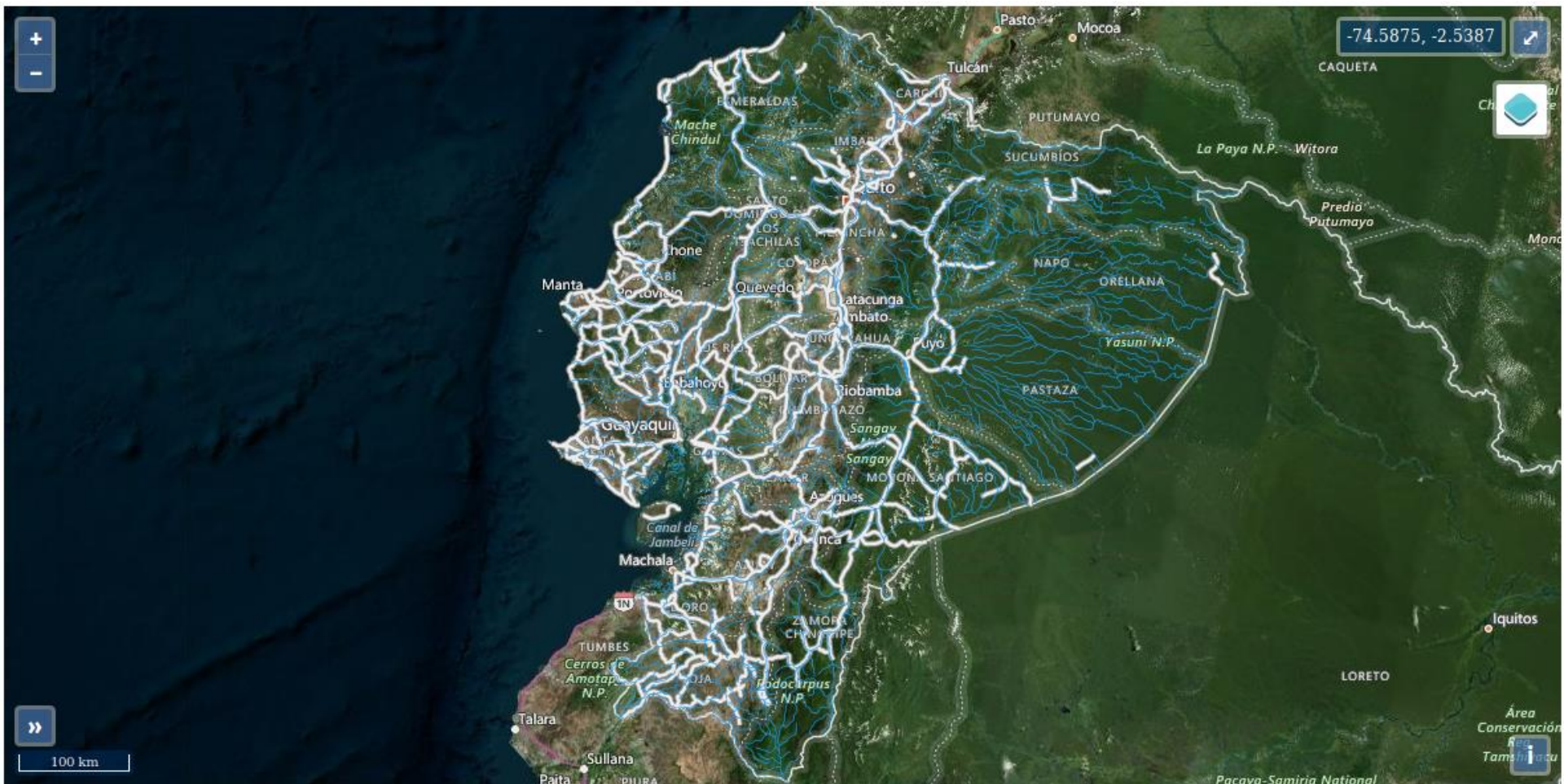
Adding Controls to the Map



Adding Controls to the Map



Adding Controls to the Map



Adding a WFS Layer

- ✓ It is required to make an **AJAX** (Asynchronous JavaScript and XML) request to get the features of a layer from GeoServer. jQuery offers a simple way to make an AJAX request. To be able to use jQuery it is necessary to download (<https://jquery.com/download/>) and import it. Use **chmod** command if you get HTTP 403 error:

```
chmod 755 /var/www/html/geo4d-openlayers/libraries/jquery-3.3.1.min.js
```

- ✓ Instead you can also use the jQuery CDN:

```
<script src=https://code.jquery.com/jquery-3.3.1.min.js integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>
```

- ✓ AJAX is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications. With AJAX, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page.
- ✓ **jQuery** (cross-platform, free and open source) is a JavaScript library designed to simplify the client-side scripting of HTML (navigate a document, select DOM elements, create animations, handle events, and develop AJAX applications).



GeoJSON and JSONP

- ✓ GeoJSON is an open standard format for encoding collections of simple geographical features along with their non-spatial attributes using JavaScript Object Notation (JSON). The features include points, line strings, polygons and collections of these types.
- ✓ JSONP (JSON with Padding) is a technique used by web developers to overcome the cross-domain restrictions imposed by browsers' same origin policy that limits access to resources retrieved from origins other than the one the page is served by.
- ✓ JSONP must be enabled in GeoServer. For that comment out the following lines in the “/usr/local/lib/geoserver-2.10.4/webapps/geoserver/WEB-INF/web.xml” file and restart GeoServer.

```
<context-param>  
  <param-name>ENABLE_JSONP</param-name>  
  <param-value>true</param-value>  
</context-param>
```



Adding a WFS Layer

- ✓ WFS layer in JSONP format is defined in the following way:

```
var vectorSource = new ol.source.Vector({
  loader: function(extent, resolution, projection) {
    var url = 'http://localhost:8082/geoserver/Ecuador/ows?service=WFS&' +
      'version=2.0.0&request=GetFeature&typeName=Ecuador:ECU_rails&' +
      'outputFormat=text/javascript&srsname=EPSG:3857&' +
      'format_options=callback:loadFeatures';
    $.ajax({url: url, dataType: 'jsonp'});
  }
});

var geojsonFormat = new ol.format.GeoJSON();
function loadFeatures(response) {
  vectorSource.addFeatures(geojsonFormat.readFeatures(response));
}

var ecuadorRailways = new ol.layer.Vector({
  title: 'Ecuador railways',
  source: vectorSource,
  style: new ol.style.Style({
    stroke: new ol.style.Stroke({
      color: 'rgb(58, 255, 81)',
      width: 4
    })
  })
});
```



Adding a WFS Layer

- ✓ `ol.source.Vector` provides a source of features for vector layers.
- ✓ `loader` is a function used to load features from a remote source. Note that data type is JSONP, which wraps up a JSON response into a JavaScript function and sends that back as a script to the browser to overcome same origin policy restrictions.
- ✓ `ol.format.GeoJSON` is a format for reading and writing data in GeoJSON format.
- ✓ `loadFeatures` is the callback function of the request defined by the URL inside the loader.
- ✓ `ol.layer.Vector` is the vector data that is rendered on the client-side.
- ✓ Add the vector layer to the layers array inside the map definition.
- ✓ jQuery AJAX documentation: <http://api.jquery.com/jquery.ajax/>.



Adding a WFS Layer

- ✓ The WFS layer on the map, the AJAX request made and the response received from the server can be seen below.

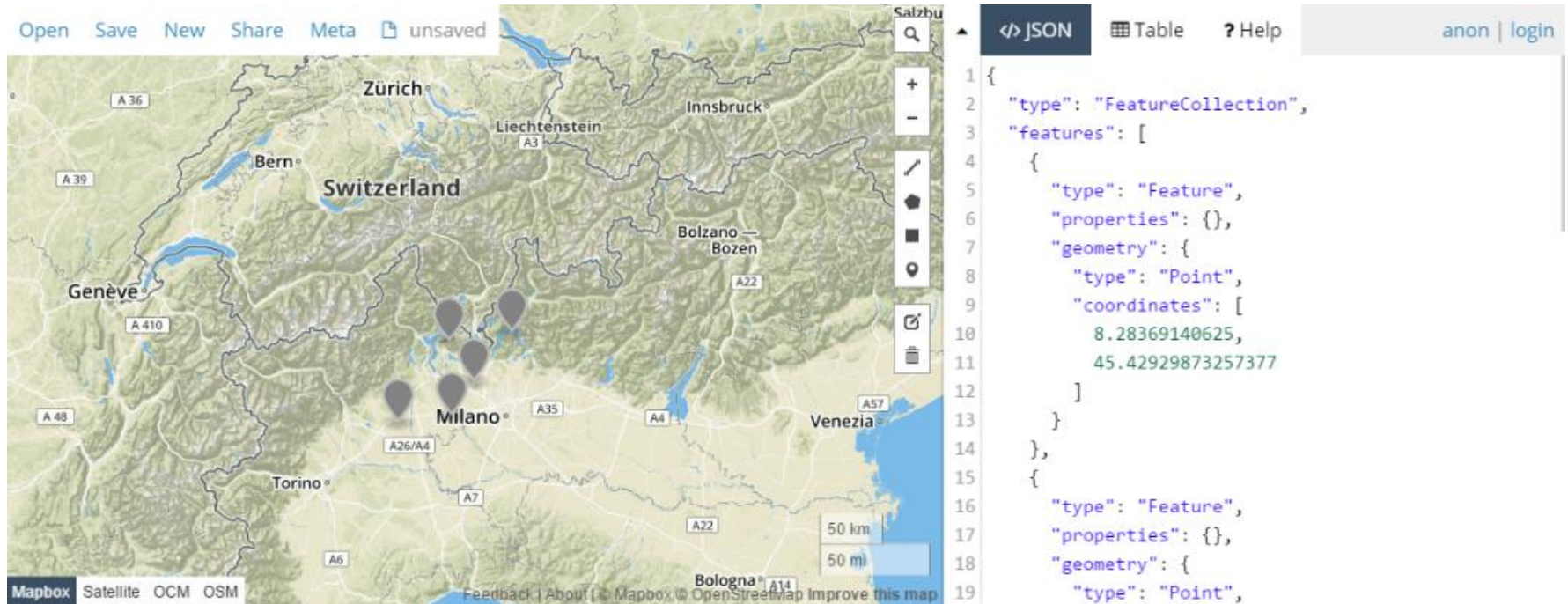
The screenshot displays a web application interface for a map of Ecuador. The map shows various geographical features, including roads, rivers, and provinces. A legend on the right side, titled "Overlay Layers", lists several layers: "Ecuador railways" (checked), "Ecuador rivers" (checked), "Ecuador roads" (unchecked), "Ecuador provinces" (checked), and "Ecuador boundary" (checked). Below the legend, there is a "Basemaps" section with options: "OpenStreetMap" (selected), "Bing Maps - Roads", "Bing Maps - Aerial", "Bing Maps - Aerial with Labels", "Stamen Watercolor", and "Stamen Toner".

At the bottom of the screen, a network inspector is open, showing a list of requests. The selected request is a GET request to the URL: `http://localhost:8082/geoserver/Ecuador/ows?service=WFS&version=2.0.0&request=GetFeature&typeName=Ecuador:ECU_rails&outputFormat=text/javascript&srsname=EPSG:3857&format_options=callback:loadFeatures&callback=jQuery33106452574348208638_1526317126135&_1526317126136`. The response payload is shown as: `LoadFeatures({"type": "FeatureCollection", "totalFeatures": 5, "features": [{"type": "Fe`.



GeoJSON

- ✓ Open the link <http://geojson.io/>.
- ✓ Add point, line and polygon features to the map and see how the GeoJSON is created.
- ✓ It is possible to export the created features in various formats, including GeoJSON, TopoJSON, KML, CSV, Shapefile using this website.



The screenshot displays the geojson.io web application. On the left, a map of Central Europe is shown with several dark purple location pins placed around Milan, Italy. The map includes labels for cities like Zürich, Bern, Geneva, Torino, and Venezia, as well as geographical features like Lake Geneva and Lake Como. The interface includes a top menu with 'Open', 'Save', 'New', 'Share', 'Meta', and 'unsaved'. On the right side of the map, there is a toolbar with icons for adding points, lines, and polygons. To the right of the map, a panel shows the generated GeoJSON code in a text editor. The code is as follows:

```
1 {
2   "type": "FeatureCollection",
3   "features": [
4     {
5       "type": "Feature",
6       "properties": {},
7       "geometry": {
8         "type": "Point",
9         "coordinates": [
10          8.28369140625,
11          45.42929873257377
12        ]
13      }
14    },
15    {
16      "type": "Feature",
17      "properties": {},
18      "geometry": {
19        "type": "Point",
```



Adding a Popup

- ✓ The popup must be contained within a “div”. So, firstly HTML code should be altered as follows:

```
<div id="map">  
  <div id="popup"></div>  
</div>
```

- ✓ By using the `getElementById` method, the defined div is accessed via its id (popup):

```
var elementPopup = document.getElementById('popup');
```

- ✓ It is also necessary to define the popup itself as an overlay, an element to be displayed over the map and attached to a single map location. Unlike controls, they are not in a fixed position on the screen, but are tied to a geographical coordinate, so panning the map will move an overlay but not a control. To define the overlay add the following lines after map definition:

```
var popup = new ol.Overlay({  
  element: elementPopup  
});  
map.addOverlay(popup);
```



Adding a Popup

- ✓ The popup itself is created using [Bootstrap](#). OpenLayers has the Bootstrap toolkit inside. You can add the libraries as follows:

```
<link rel="stylesheet"
href="libraries/ol_v4.6.5/apidoc/styles/bootstrap.min.css" type="text/css">
<script src="libraries/ol_v4.6.5/apidoc/scripts/bootstrap.min.js"></script>
```

- ✓ Now let's add the event (click) listener for the map to create the popup.

```
map.on('click', function(event) {
    var feature = map.forEachFeatureAtPixel(event.pixel, function(feature, layer) {
        return feature;
    });

    if (feature != null) {
        var pixel = event.pixel;
        var coord = map.getCoordinateFromPixel(pixel);
        popup.setPosition(coord);
        $(elementPopup).attr('title', 'Ecuador railways');
        $(elementPopup).attr('data-content', '<b>Id: </b>' + feature.get('FID_rail_d') +
            '<br><b>Description: </b>' + feature.get('F_CODE_DES'));
        $(elementPopup).popover({'placement': 'top', 'html': true});
        $(elementPopup).popover('show');
    }
});
```



Adding a Popup

- ✓ Moreover we can add another event (**pointermove**) listener to change the icon of the cursor. The event listener is as follows:

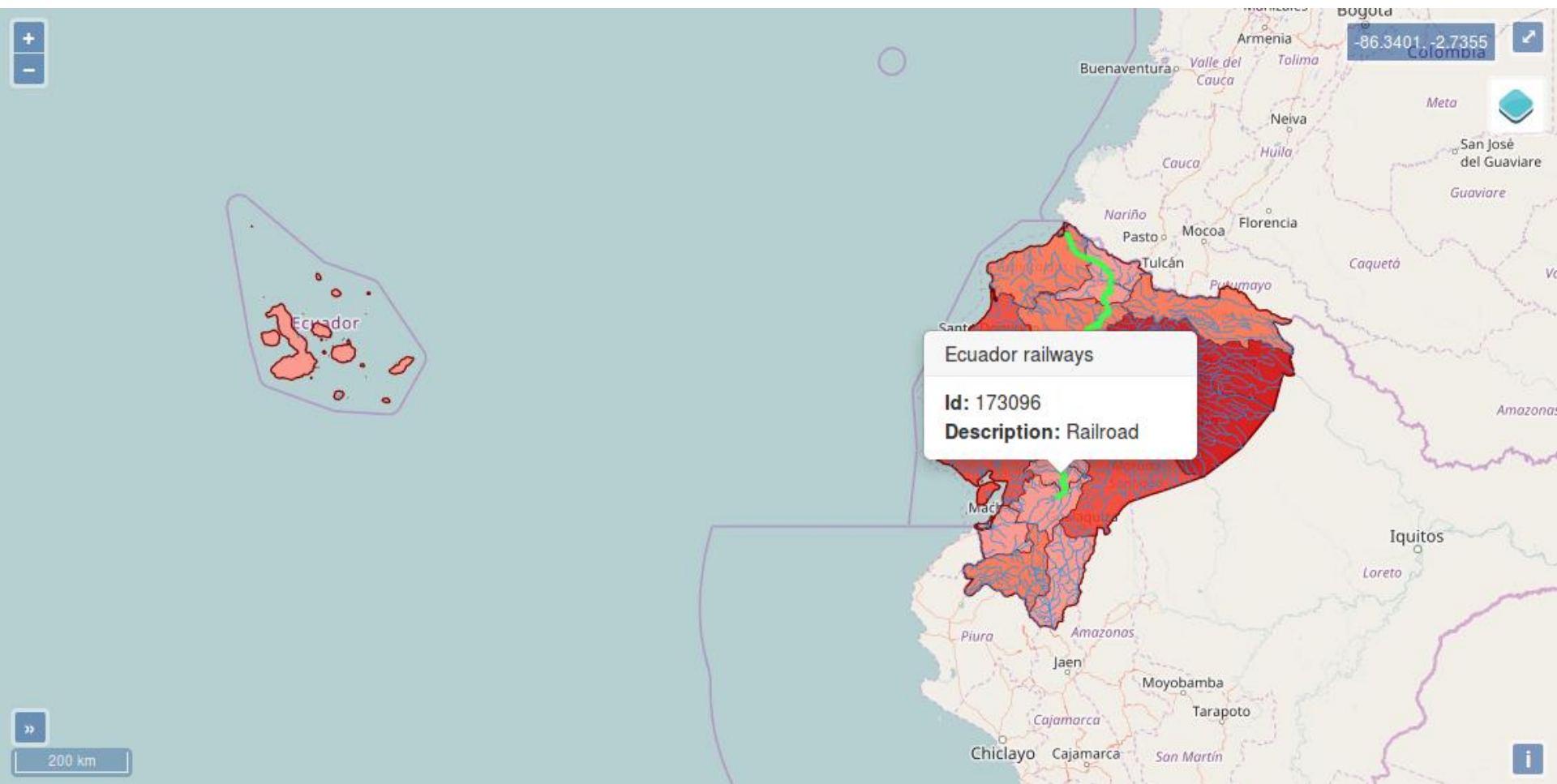
```
map.on('pointermove', function(e) {  
    if (e.dragging) {  
        $(elementPopup).popover('destroy');  
        return;  
    }  
    var pixel = map.getEventPixel(e.originalEvent);  
    var hit = map.hasFeatureAtPixel(pixel);  
    map.getTarget().style.cursor = hit ? 'pointer' : '';  
});
```

- ✓ **pointermove** is triggered when a pointer is moved.
- ✓ Method **getEventPixel** returns the map pixel position for a browser event relative to the viewport.
- ✓ Method **hasFeatureAtPixel** detects if features intersect the pixel on the viewport.
- ✓ Lastly it is possible to set the width of the popup with the following CSS rule:

```
.popover {  
    width: 200px;  
    height: auto;  
}
```



Adding a Popup



WMS GetFeatureInfo

- ✓ First let's create the `<div>` for GetFeatureInfo, and give it an id `get-feature-info`.

```
<div id="get-feature-info"></div>
```

- ✓ Add the following event listener to the map for the click event:

```
map.on('click', function(event) {  
    document.getElementById('get-feature-info').innerHTML = '';  
    var viewResolution = (map.getView().getResolution());  
    var url = ECU_roads.getSource().getGetFeatureInfoUrl(event.coordinate,  
        viewResolution, 'EPSG:3857', {'INFO_FORMAT': 'text/html'});  
    if (url)  
        document.getElementById('get-feature-info').innerHTML = '<iframe  
seamless src="' + url + '"></iframe>';  
});
```



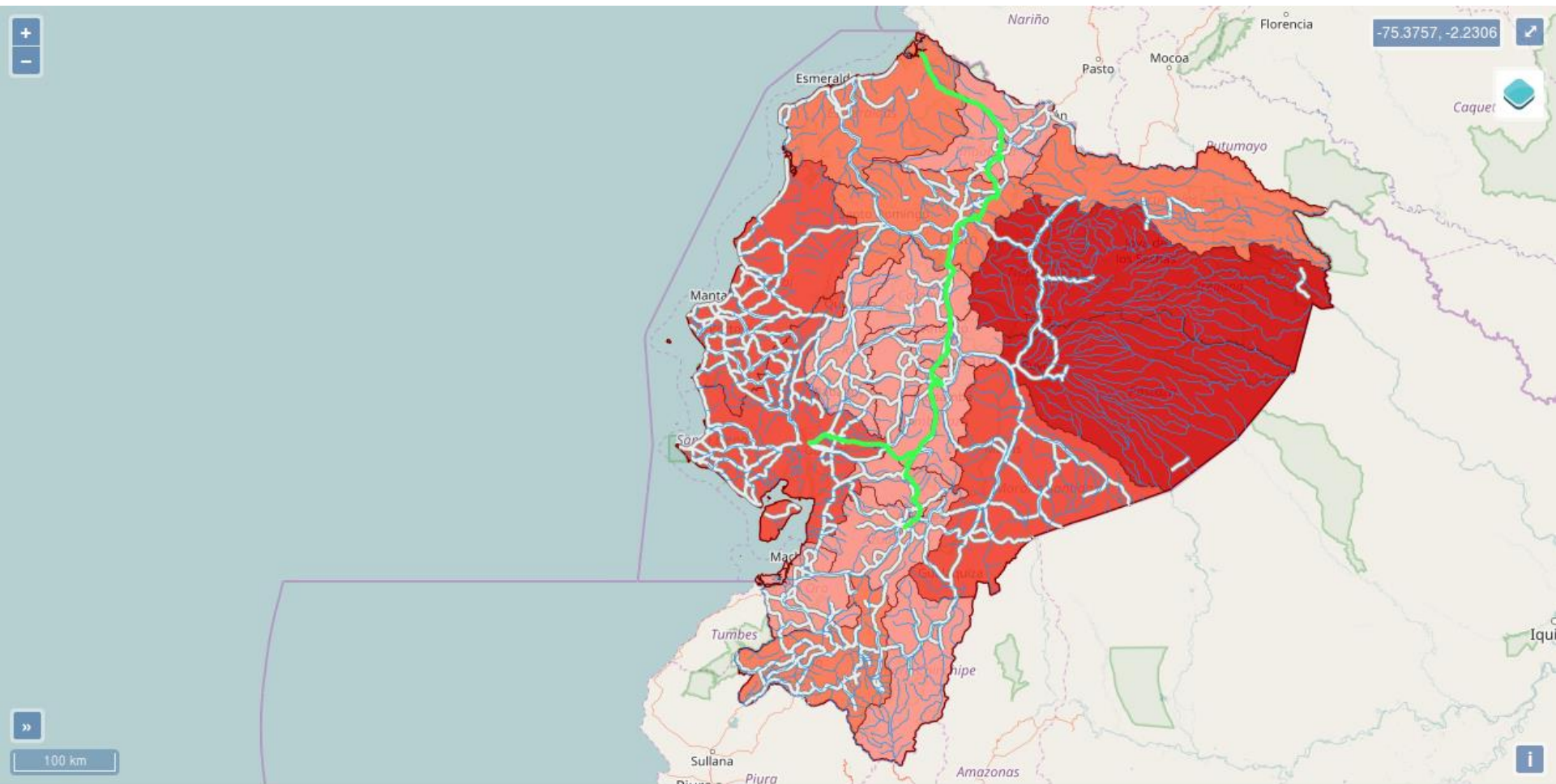
WMS GetFeatureInfo

- ✓ The **innerHTML** property sets or returns the HTML content (inner HTML) of an element.
- ✓ The **getSource** returns the associated source of the image layer. **getFeatureInfoUrl** returns the GetFeatureInfo URL for the passed coordinate, resolution, and projection. Returns undefined if the GetFeatureInfo URL cannot be constructed.
- ✓ If GetFeatureInfo URL is successfully constructed, the URL is placed inside the <iframe> (HTML inline frame element), which is used to display a web page within a web page.
- ✓ To improve the look of the iframe add the following CSS code:

```
iframe {  
  border: 0px;  
  height: 110px;  
  width: 90vw;  
}
```



WMS GetFeatureInfo



ECU_roads

fid	MED_DESCR	RTT_DESCR	F_CODE_DES	ISO	ISOCOUNTRY
ECU_roads.304	Without Median	Primary Route	Road	ECU	ECUADOR



Adding GeoTIFF

- ✓ As GeoTIFF is a raster format, a GeoTIFF layer can be added to the map using **Image** layer object. The layer (**Ecuador:Ecuador_tourism_map_georeferenced**) is defined as follows:

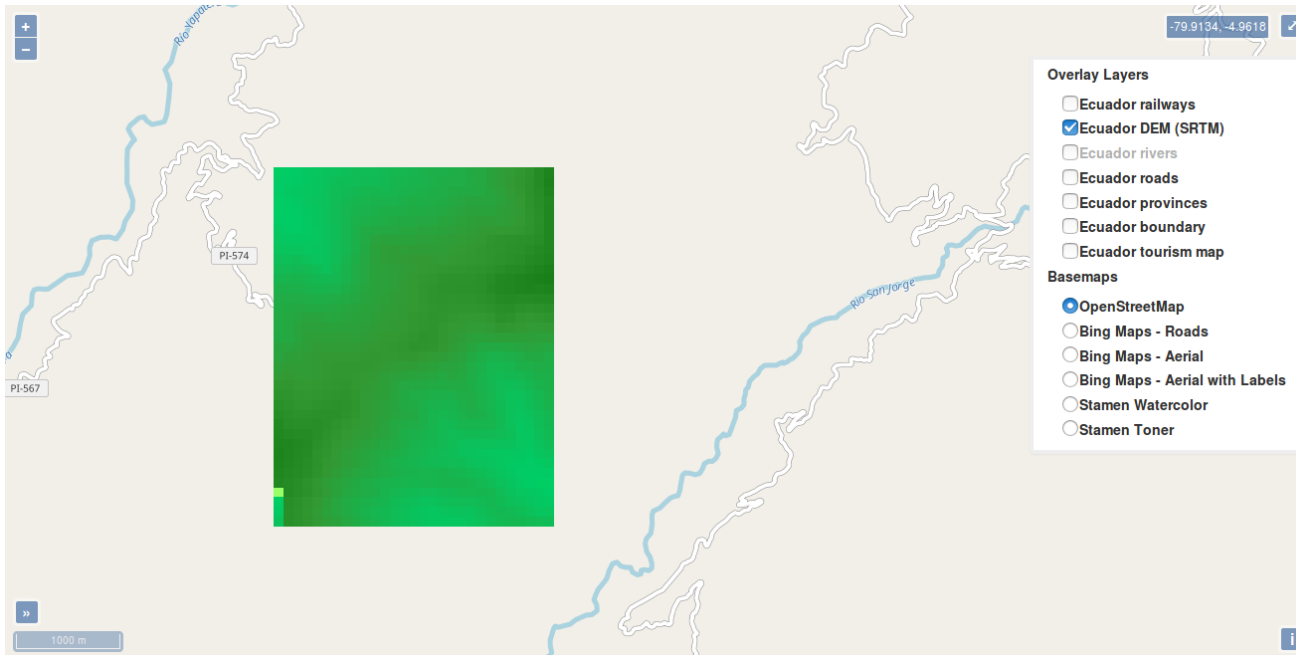
```
var ecuadorTourismMap = new ol.layer.Image({  
  title: 'Ecuador tourism map',  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:Ecuador_tourism_map_georeferenced'}  
  })  
});
```



Adding an ArcGRID Layer

- ✓ As ArcGRID is a raster format, a layer in this format can be added to the map using **Image** layer object. The layer (**Ecuador:SRTM_Ecuador**) is defined as follows:

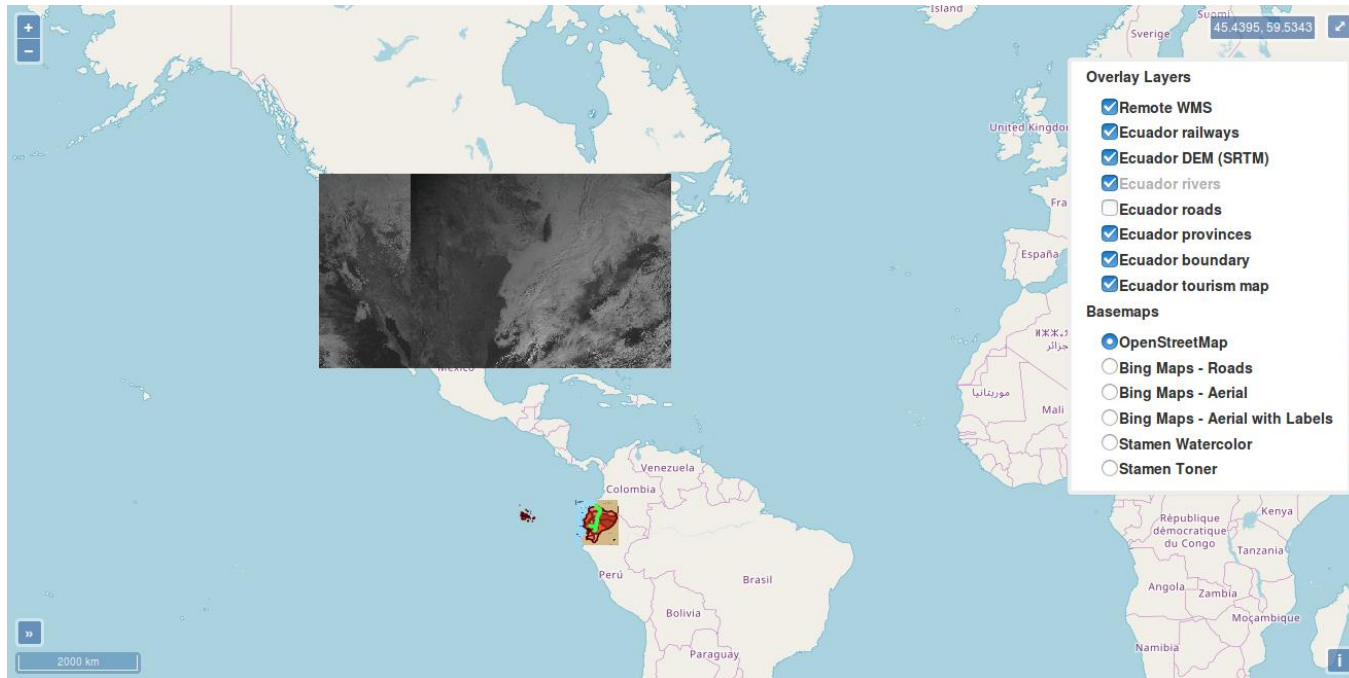
```
var ecuadorSrtm = new ol.layer.Image({  
  title: 'Ecuador DEM (SRTM)',  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:SRTM_Ecuador'}  
  })  
});
```



Adding Remote WMS

- ✓ In the same manner a remote WMS layer published in GeoServer can be added to the map. The layer (**Ecuador:conus_vis_4km**) is defined as follows:

```
var remoteWms = new ol.layer.Image({  
  title: 'Remote WMS',  
  source: new ol.source.ImageWMS({  
    url: 'http://localhost:8082/geoserver/wms',  
    params: {'LAYERS': 'Ecuador:conus_vis_4km'}  
  })  
});
```



Exercise

- ✓ Add the following basemaps, and in the following order to the map:

OpenStreetMap

Bing Maps – Roads

Bing Maps – Aerial

Bing Maps – Aerial with Labels

Stamen Watercolor

Stamen Toner



Exercise

- ✓ Add the following overlay layers published in GeoServer and in the following order to the map:
- ✓ **ECU_tourism_map_georeferenced** (Ecuador tourism map) as WMS layer
- ECU_adm0** (Ecuador national boundary) as WMS layer
- ECU_adm1** (Ecuador provinces) as WMS layer
- ECU_adm2** (Ecuador cantons) as WMS layer
- ECU_adm3** (Ecuador parishes) as WMS layer
- ECU_water_areas_dcw** (Ecuador lakes) as WMS layer
- ECU_water_lines** (Ecuador rivers) as WMS layer
- ECU_roads** (Ecuador roads) as WFS layer
- ECU_rails** (Ecuador railways) as WMS layer
- ✓ Use the default styles (specified during the GeoServer practices) for all the WMS layers with the exception of:
 - ECU_adm2** (Ecuador cantons) → use the **grass** style



Exercise

- ✓ For the WFS layer **ECU_roads**, use black and dashed line with width 2.
- ✓ Make the following layer visible only at resolution > 1:2000 **ECU_water_lines_dcw**.
- ✓ Make the following layers not visible by default when the web GIS is opened:
ECU_tourism_map_georeferenced (Ecuador tourism map) **ECU_water_areas_dcw** (Ecuador lakes).
- ✓ Include all the controls (i.e. scale line, full screen, overview map, mouse position, layer switcher) added during the practice, together with their modified styles.
- ✓ Include popup for the WFS layer **ECU_roads** and display all of its attributes' values inside the popup.
- ✓ Implement GetFeatureInfo for **ECU_water_areas_dcw** layer.



References

- ✓ Web mapping client comparison: <http://geotux.tuxfamily.org/index.php/en/geoblogs/item/291-comparacion-clientes-web-v6>
- ✓ OpenLayers official website: <https://openlayers.org/>
- ✓ OpenLayers 4.6.5 user documentation: <https://openlayers.org/en/v4.6.5/doc/>
OpenLayers 4.6.5 API (application programming interface):
<https://openlayers.org/en/v4.6.5/apidoc/>
- ✓ OpenLayers 4.6.5 example gallery: <https://openlayers.org/en/v4.6.5/examples/>
- ✓ Boundless OpenLayers Workshop: <http://workshops.boundlessgeo.com/openlayers3/>
- ✓ Marco Minghini, OpenLayers Basics, Geographic Information Systems (GIS) course, Politecnico di Milano - Como Campus, academic year 2013-2014.





POLITECNICO
MILANO 1863

Candan Eylül Kilsedar

candaneylul.kilsedar@polimi.it

**Department of Civil and Environmental Engineering, Politecnico di Milano, Piazza
Leonardo da Vinci 32, 20133 Milano, Italy**



<https://creativecommons.org/licenses/by-sa/4.0/legalcode>