Make your own GIS service

Summer School on Digital Humanities

Course material available at

https://bitbucket.org/augusto_ciuffoletti/digitalmaps4ssdh

Augusto Ciuffoletti

21 maggio 2021

Make your own GIS service

- We have seen how open services provide many functionalities to
  - produce a *live* map, with web links and multimedia contents
  - share it with other
  - export the data across tools and other services
- However it may be the case that what we want a service that does not exactly match one of those that are already avaiable
- In that case we need to code our own web service
- The task is simplified by the existence of a powerful open source library, *leaflet*
- In this part of our tutorial we scratch the surface of this tool to understand its potential

Stackblitz

- The tool we are going to use to practice the *leaflet* library is Stackblitz (`https://stackblitz.com/`)
  - First step, create an account on Stackblitz
- Stackblitz is an online IDE for JavaScript
- Once you log-in it presents you an assortment of different JavaScript platforms
  - e.g. Angular, Vue, React, and also plain JavaScript
- Click on the yellow JavaScript icon and wait until the project is created
- The screen is divided into three columns, a two toolbars on the top and the left side of the window

Stackblitz interface

- In the right column we see the preview of our service
  - the URL on top of the screen is functional, try it...)
- In the center column you have your code
  - Try to change the string in line 6 and notice the preview change
- In the left column you have the project content and github reference
- The left toolbar controls the content of the left column
- The top toolbar is for project management

LAB: take you time to try the above steps and look around the content and the controls

Project lifecycle

- The project code resides on the Web server
- The user obtains the code with a HTTP GET on the browser (i.e., visiting the URL)
- The the browser runs the JavaScript code and displays the result
- The user interacts with the browser display by clicking buttons, filling forms and such
- All this is synthesised in the StackBlitz screen, but the Web server is useful only for development
- You can host a real deployment on Firebase (third icon in the left toolbar)
  - free plan available, Google account needed

Step by step *Leaflet* tutorial

- Use the project create in the above step
- Click on the *Connect repository* button
  - do not fill the New Repository field
- Select *import an existing repository*
  - ...and ignore the yellow warning
- Copy the following URL in the *Paste GitHub repository URL* field
  - https://github.com/AugustoCiuffoletti/leafletExercise
- Next click *Import repository*
- Wait for the project to load

Moving across the tutorial

- Observe the top of the left toolbar
- Below the line with the name of the GitHub repository (`AugustoCiuffoletti/leafletExample`) you should see a line headed by the *branch* icon
- Use that line to navigate through the tutorial steps
- The line should show `master` presently
- Click on it to see the list of the tutorials steps
  - in case, now select `master`
  - to move to a different step just click on it
- In the README.md file you find further comments

Step 0: the background raster

- The first step in our tutorial consists of using the *Leaflet* library in order to display an OpenStreetMap raster
- To this end we include in the HTML file:
  - the css and the library *Leaflet* in the `head` element
  - a `div` element with the identifier of the element for map display
- The index.js file contains the JavaScript code.
- The capital `L` stands for the *Leaflet*class
- So we apply the global method `map` which takes two parameters
  - the id of the element hosting the raster
  - A JavaScript object that configures the map display.
  - We opted for a minimalistic configuration with:
    - the position of map center (the Pisa tower)
    - the zoom level
    - the source of the raster (OpenStreetMaps)

Step 0: Lab activity

- Browse the web to find the coordinates of a place at your choice as the center of the raster
- Modify/remove the zoom factor

IMPORTANT:

- you **cannot** commit your updates on my repo (Error 403)
- you can *Save* your updates,
  - but you will loose your work when you switch branch (Discard Changes)
- you can undo updates with Ctrl-z
- you can *Fork* a branch and then *Connect a repository* of your own
  - this works on a single branch
- you can clone the whole repository (all branches) in your computer and push it on a new repo

Step 1: a decorated feature

- We want to draw a point corresponding to the coordinates of the leaning tower truncated at two digits
  - and attach to the point an explanatory popup
- We define variables for the latitude and the longitude of the tower
  - and another two with the truncated values
- The new point is defined (constructed) as an object in the class *Marker* using the computed coordinates
- We apply the *bindPopup* method to the marker to attach the popup
- The *addTo* method is applied to the marker in order to include it in the map, which is the method parameter
- Finally *openPopup* is applied to the marker to display the attached popup

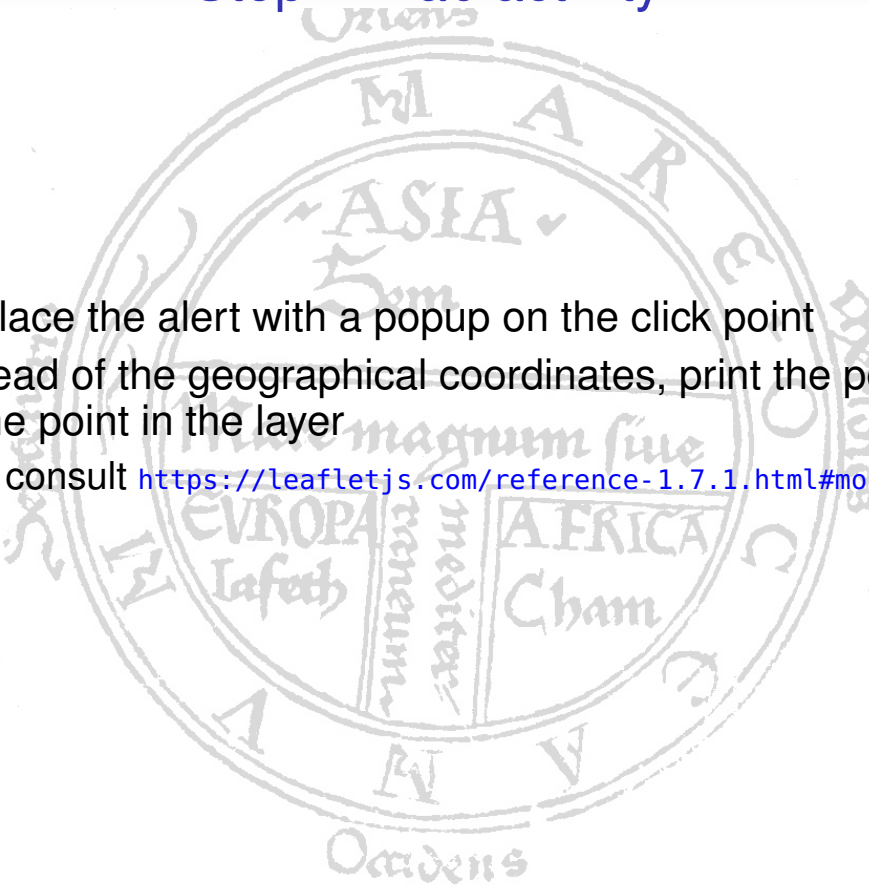
Step 1: Lab activity

- Try to truncate at three and four digits, or to place the marker on the center of the map
- Place the marker in a different place and change the message

Step 2: show the coordinates

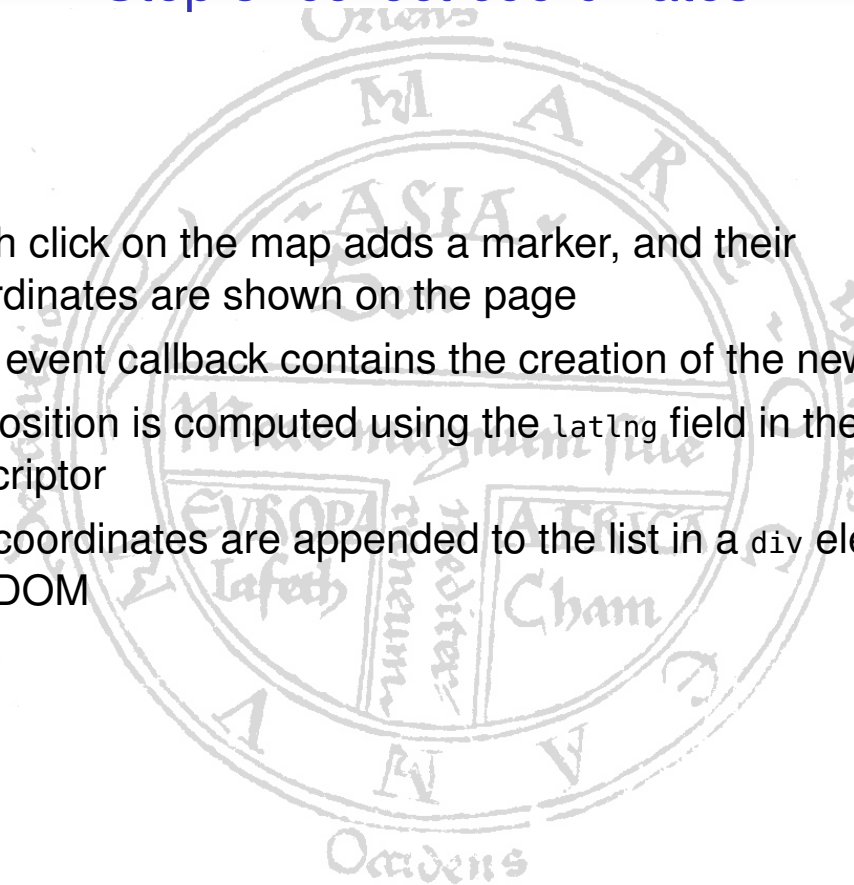
- When the user clicks on the map an alert appears with the coordinates of the click
- We apply the *on* method to the map to catch events
  - the first parameter is the name of the event we want to capture
  - the second parameter is a callback that takes the event description as a parameter
  - the callback displays the alert that contains data extracted from the event descriptor `e`
  - the event descriptor is an object
  - we extract the `lat` and `lng` fields in the `latlng` field.

Step 2: Lab activity



- Replace the alert with a popup on the click point
- Instead of the geographical coordinates, print the position of the point in the layer
  - consult https://leafletjs.com/reference-1.7.1.html#mouseevent

Step 3: collect coordinates



- Each click on the map adds a marker, and their coordinates are shown on the page
- The event callback contains the creation of the new marker
- its position is computed using the `latlng` field in the event descriptor
- the coordinates are appended to the list in a `div` element of the DOM

Step 3: Lab activity

- Display the distance of the point from the center instead of its coordinates
  - consult https://leafletjs.com/reference-1.7.1.html#map-conversion-methods

Step 4: enumerated markers

- An progressive index is assigned to each new point
- The index is shown in the list and added as a *title* field in the marker definition
  - the title field is automatically displayed when the mouse hovers on the marker
- We add a new global variable `n`
- The event callback increments the variable each time it is run
- The value of `n` is displayed on each line in the list
- The marker constructor now takes a second parameter containing the marker options
  - among which the `title` option

Step 4: Lab activity

- Configure the marker as draggable (discard that the displayed is not consistent anymore)
  - consult https://leafletjs.com/reference-1.7.1.html#marker
- (advanced) show the coordinates inside the *title* and update them when the marker is dragged
  - consult the same manual page of the previous lab activity

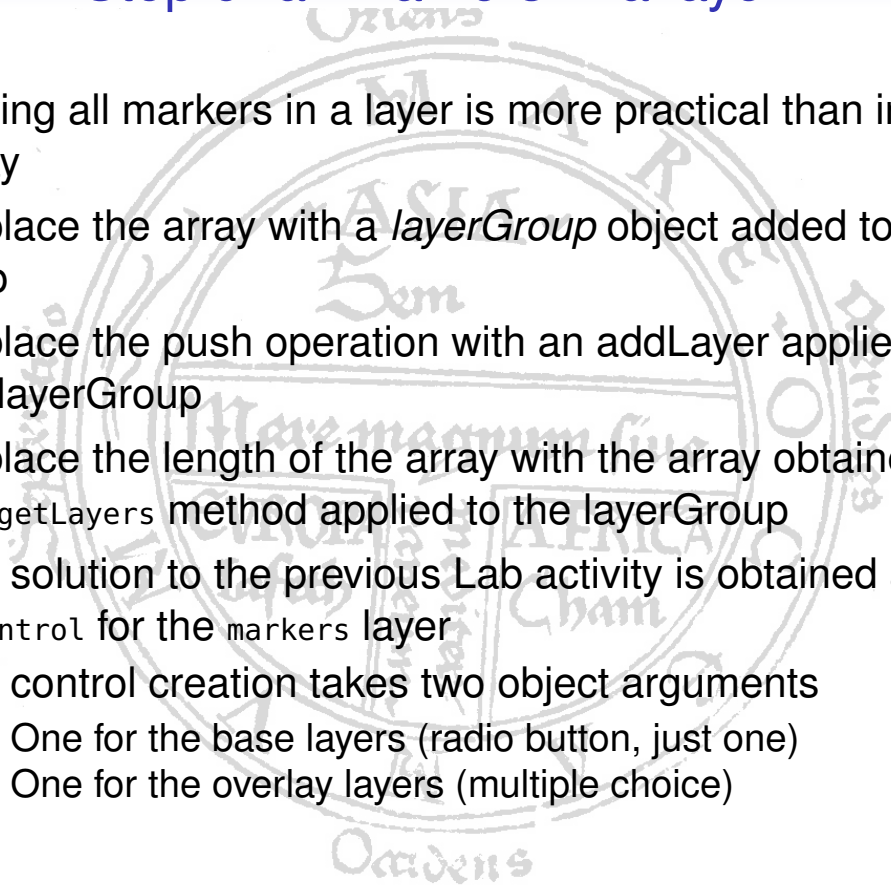
Step 5: all markers in a row

- Record the markers in an array to have them accessible
  - in the previous steps the marker was a local variable in the callback
- Create an array for the markers
- Push markers in the array
- `n` index corresponds to array length

Step 5: Lab activity

- Create a button that hides all the markers
  - Use opacity, same manual page of the previous lab activity

Step 6: all markers in a layer



- Having all markers in a layer is more practical than in an array
- Replace the array with a *layerGroup* object added to the map
- Replace the push operation with an addLayer applied to the layerGroup
- Replace the length of the array with the array obtained with the `getLayers` method applied to the layerGroup
- The solution to the previous Lab activity is obtained adding a `control` for the `markers` layer
- The control creation takes two object arguments
  - One for the base layers (radio button, just one)
  - One for the overlay layers (multiple choice)

Step 6: Lab activity

- Add a popup to all features in the layer
  - consult https://leafletjs.com/reference-1.7.1.html#layergroup

Step 7: GeoJSON serialization

- it is handy to have a standard string representation of a piece of data (serialization)
  - e.g. to store the data in a file
- the GeoJSON representation can be easely transformed into a JSON string
- We want to print in the console the JSON representation of our markers
- the layer @toGeoJSON method serializes a layer of markers as point features
  - alas, we will loose the *title* field
- In the logging command we serialize the content of our map

Step 8: a map in the cloud

- We want to store our markers in the cloud
- The simplest option is to use a KeyValue as a Service provider
- The string containing the serialized layer containing the markers is associated with an URI
- A button in the service interface allows the user to acquire a reserved URI (with an HTTP post)
  - after that the GeoJSON string is associated to the URI (with another HTTP POST)
- The `toGeoJSON` method converts the markers layer into a GeoJSON JavaScript object
- The `stringify` method serializes the object as a String object

Step 9: retrieving the markers

- Now we want to be able to retrieve the markers from a key-value service URI indicated by the user
- The user pastes the URI in the appropriate box
- The layer is cleared from previous markers
- The JSON string is downloaded using the provided URI and
-

Firebase deployment

- For this you need a further account, unless you already have a Google account

Time for a uMap *'challenge'* ! (step 1/3)

Geolocate yourself

- Insert in the map **http://u.osmfr.org/m/151488** a feature *point* corresponding to a place you like. Any: residence, domicile, birth, your last hike etc... Put your nickname in the *'description'* of the point and make it visible.

Time for a uMap *'challenge'* (step 2/3)

The map for the tower

- Invite a friend that arrives by train to see the Tower. Create a map with a walking path to the bus stop, then the one by bus, and from the stop to the tower. At the bus stop, enter a description indicating the bus number and direction, the stop name and the estimated travel time at the stop. You can use fantasy buses. Finally, enter the short URL in the description of the *point* of the previous step.

Time for a uMap *'challenge'* (step 3/3)

Track your walk

- select a nearby point of interest (e.g. a bridge, a pub, or a tower), locate it on Gaia GPS map, reach the point recording a track with Gaia GPS, then take a picture. Go back to your PC, export the track as a GPX file in Gaia GPS cloud and import in uMap using your computer. Find a way to link your picture in waypoint description.