Make your own GIS service

Summer School on Digital Humanities

Course material available at

https://github.com/AugustoCiuffoletti/DHSS_2025

Augusto Ciuffoletti

31 marzo 2025

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 others
  - export the data across tools and other services
- However we may have requirements that do not exactly match an existing service
- 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 concluding tutorial we scratch the surface of this tool to understand its potential

How to proceed

- The tool we are going to use to practice the *leaflet* library is Stackblitz (https://stackblitz.com/)
- Stackblitz is an online IDE for JavaScript
- The tutorial will present the step-by-step creation of a simple app that:
  - Displays a map
  - Allow the user to add markers to the map
  - Exports the markers as a GeoJSON string
  - Stores the layer in the cloud (until September 2025)
- The code for each step can be viewed, tested, and modified in a Stackblitz project
- The link to each project is in the title of the slide

Stackblitz interface

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

One screen development environment

- 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 browser runs the JavaScript code and displays the result
- The user interacts with the browser clicking buttons, filling forms and such
- All this is synthesised in the StackBlitz screen
  - using a Web server which is appropriate only for development
  - a real deployment may run on Firebase (third icon in the left toolbar)
  - free plan available, Google account needed

Step by step *Leaflet* tutorial

- The support material for this tutorial runs on Stackblitz
- To use it, start from the project created 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 see a line headed by the *branch* icon
- Use this line to navigate through the tutorial steps
- The line should show `master` presently
- Click on it for a list of the tutorials steps
  - select `master` it is not on display
  - to move to a different step just click on it
- In the README.md file you find further comments

Step 1: the background raster (project)

- The first step in our tutorial consists of using the *Leaflet* library to display an OpenStreetMap raster
- To this end we include in the HTML file:
  - a `head` element with the CSS and the JavaScript for *Leaflet*
  - a `div` element for map display (its `id` is `aMap`)
- The index.js file contains the JavaScript code of our App
- The capital `L` stands for the *Leaflet* class
- So we apply the global method `map` which takes two parameters
  - the id of the DOM element hosting the raster
  - A JavaScript object that describes how the map is displayed
  - We opted for a minimal description with only:
    - the position of map center (the Pisa tower)
    - the zoom level
    - the source of the raster (OpenStreetMaps)

Step 1: 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 *Connect a repository* of your own on GitHub (recommened)
- 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
  - 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 2: show the coordinates (project)

- 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 *click* 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 an alert containing 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 (project)

- 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 (project)

- 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 (ignore that the displayed coordinates become inconsistent)
  - 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 an array (project)

- 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
  - no need to increment it

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 (project)

- 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
- Compute `n` as the length of 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)
- See the effect on the layers button top-right in the map

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 (project)

- 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, and viceversa
- We want to print in the console the JSON string for our markers
- The `toGeoJSON` method converts the markers layer into a JavaScript object with the GeoJSON format
  - alas, in this way we lose the *title* field
- The `stringify` method serializes the object as a String object
- The string is finally recorded in the log

Step 7: Lab activity

- Is there any way to record the *title* field in the JSON string?
- Study the geoJSON format in the console and find a solution
- If needed see :
  - `https://geojson.org/` for geojson syntax
  - `https://leafletjs.com/reference-1.7.1.html#marker` for the toGeoJSON method

Step 8-10: a map in the cloud

- We want to store our markers in the cloud
- The simplest option is to use a Key-Value service
  - a basic one is the one I implemented on MongoDB Atlas (just demonstration, not for public use)
- A *New* button in the interface allows the user to acquire a reserved key (step 8)

(project)

- A *Save* button allows to update the cloud record (after filling the Key box) (step 9)

(project)

- A *Load* button allows to download the cloud record (after filling the Key box) (step 10)

(project)

Firebase deployment

- For this you need a Google account
- You need first to access the console of the service at `https://console.firebase.google.com/` and add a new project
  - in the following dialog, do not enable Google Analytics
  - observe the firebase logo, in the upper left corner
- In the Stackblitz window click on the firebase logo in the left toolbar
- Click on the name of your project and next "Deploy"
- Finally click on the "Open Site", or visit `<YourProjectName>.firebaseapp.com`
- Your app is now permanently available at that URL