REST Advanced Lab

Camilla Pelagalli, Daniele Parmeggiani, Giovanni Rigotti, Shandy Darma, Stefano Faccio

DISI, University of Trento, Service Design & Engineering, 2022/2023

Advanced Uses of REST APIs

What happens when your service builds on top of other services?

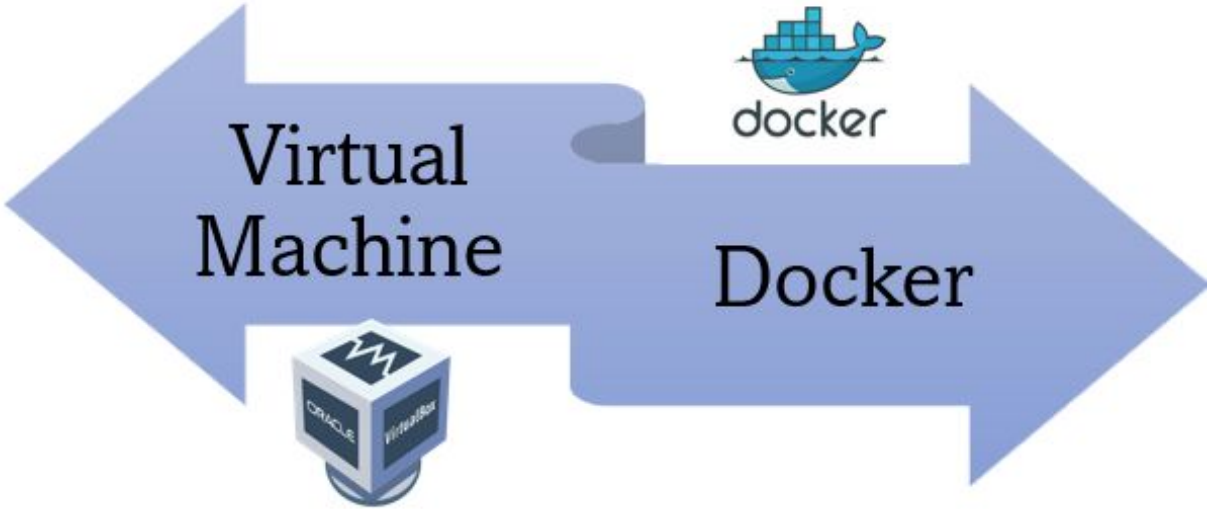
How do you interact with them?

How can they be reached?

How do you use several together?

How to run

We have provided you with a VM and a docker compose file so that you can also work on this lab without spinning up a VM.



https://github.com/dpdani/sde-rest-advanced

Intro

Today we brought a simple web service that uses Lab 4's service and we're going to:

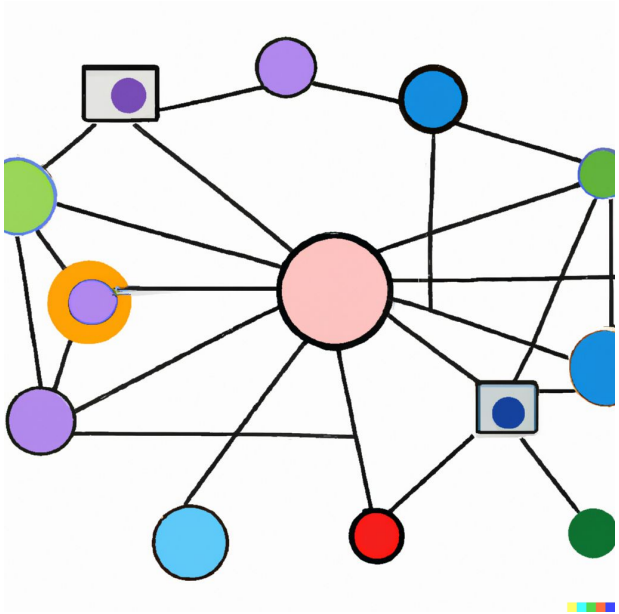
- ➔ **Explore service composition**
  With Lab 4, Nominatim, and OpenStreetMap
- ➔ **Modify a service**
  Learn how to integrate new functionalities
- ➔ **Drink coffee**
  Required for a good grade

REST Recap

Use HTTP methods (with their semantics) on “resources” provided by a service.

Resources are *identifiable* and possibly linked together.

The service is *stateless* (or at least should be), while the resources it manages have a possibly transitioning state.



Lab 4

OpenStreetMap

Nominatim

Leaflet

Services used

Lab 4

In the previous lab, we've worked on an event management service.

We've built a service called *NearMeEvents* on top of theirs to let a user look for events.

With Lab4, we could use Mobilizon data to show the users real events. In this Lab, we're going to make use the fixed data in Lab4's repo.

OpenStreetMap

OpenStreetMap is a map of the world, created by people like you and free to use under an open license.

https://www.openstreetmap.org/

Our map providers: this is where the images displayed in the frontend come from.

Nominatim

Open-source geocoding with OpenStreetMap data

https://nominatim.org/

Geocoding means searching some place's name in order to find its location on Earth.

We're going to use this service in Exercise 3.

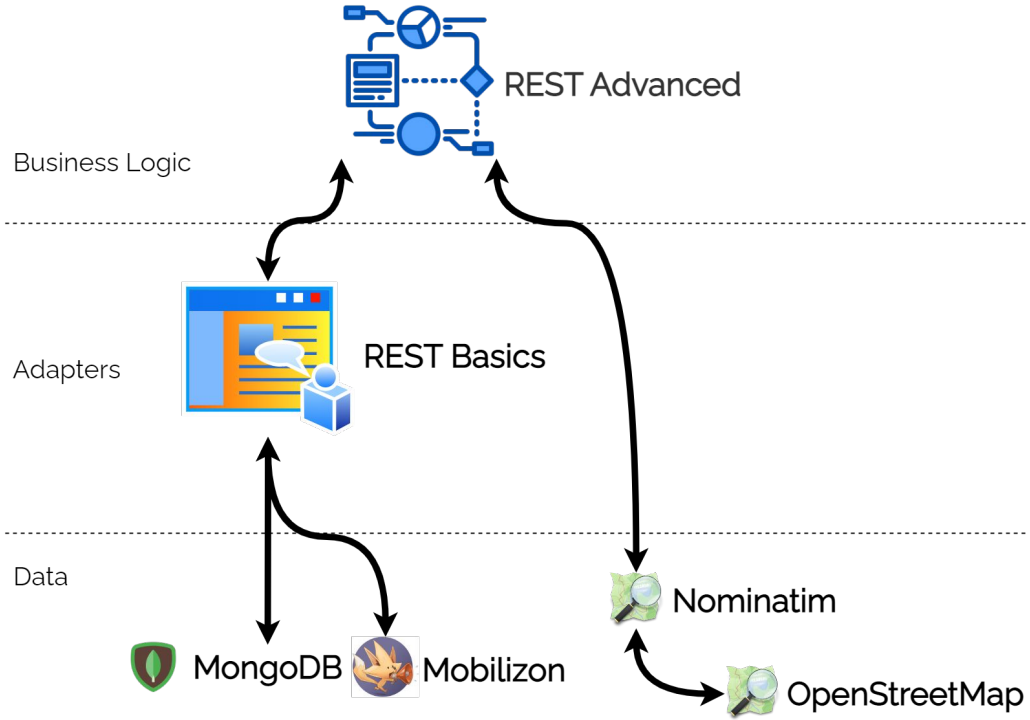
Leaflet

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps

https://leafletjs.com/

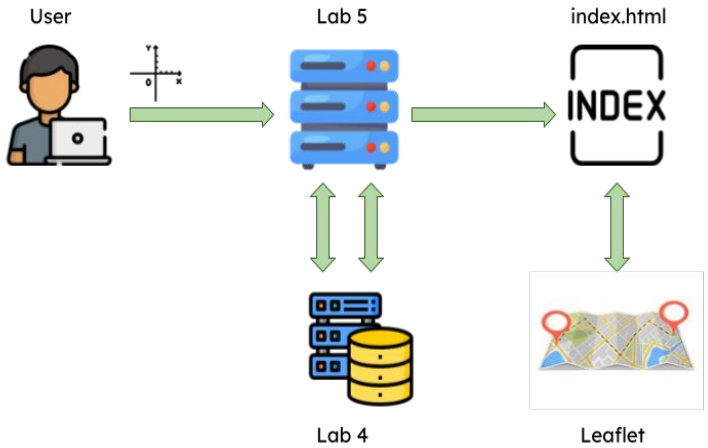
This library powers the frontend interactive maps displayed.

Logical layers



The Workflow

1. User inputs a location
2. We search Lab4 events to know which are close to the user
3. We query Lab4 to get information about each event
4. We redirect to `index.html`
5. Client-side JavaScript uses Leaflet to populate the map with the data fetched in the previous steps

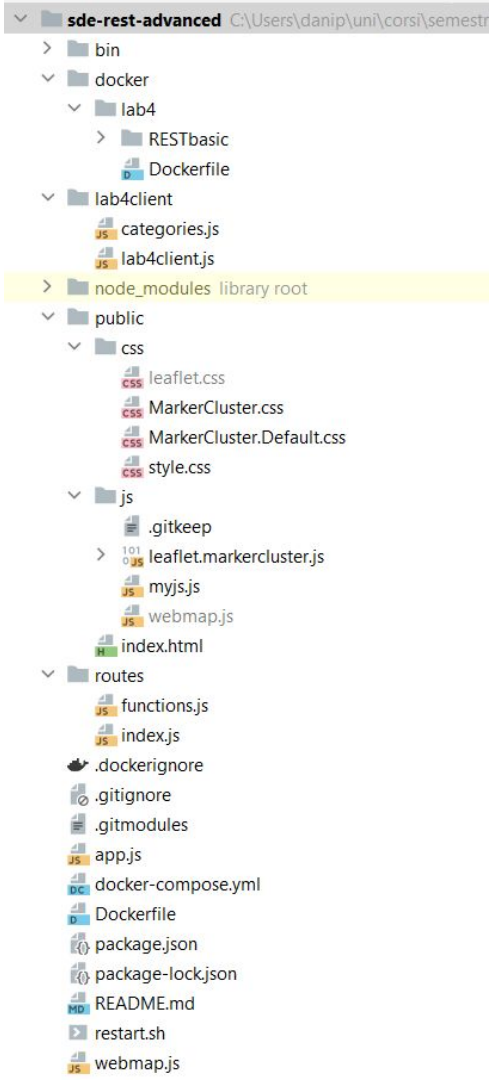


The Code

Available in the Virtual Machine in

~/Desktop/rest_advanced/sde-rest-advanced

and at
https://github.com/dpdani/sde-rest-advanced



lab4client/

- `searchEvents()` → list of IDs
- `prepareEvents()` → list of events formatted for display
- `prepareOneEvent()` → fetch one event from Lab4 and format it

routes/

Our REST endpoints.

The `/flow` endpoint implements the workflow presented before.
This is where the different services are “glued” together.

public/ & webmap.js

Static data and scripts served for the frontend.

The `webmap.js` script generates the map (Leaflet is used here).

Web Application Setup

If you start the app for the first time:

- Open a terminal in *~/Desktop/rest_advanced/sde-rest-advanced*
- Type: **docker-compose up**

If you have modified the code and just want to restart the app, type the following command:

- **docker-compose up --force-recreate --build**

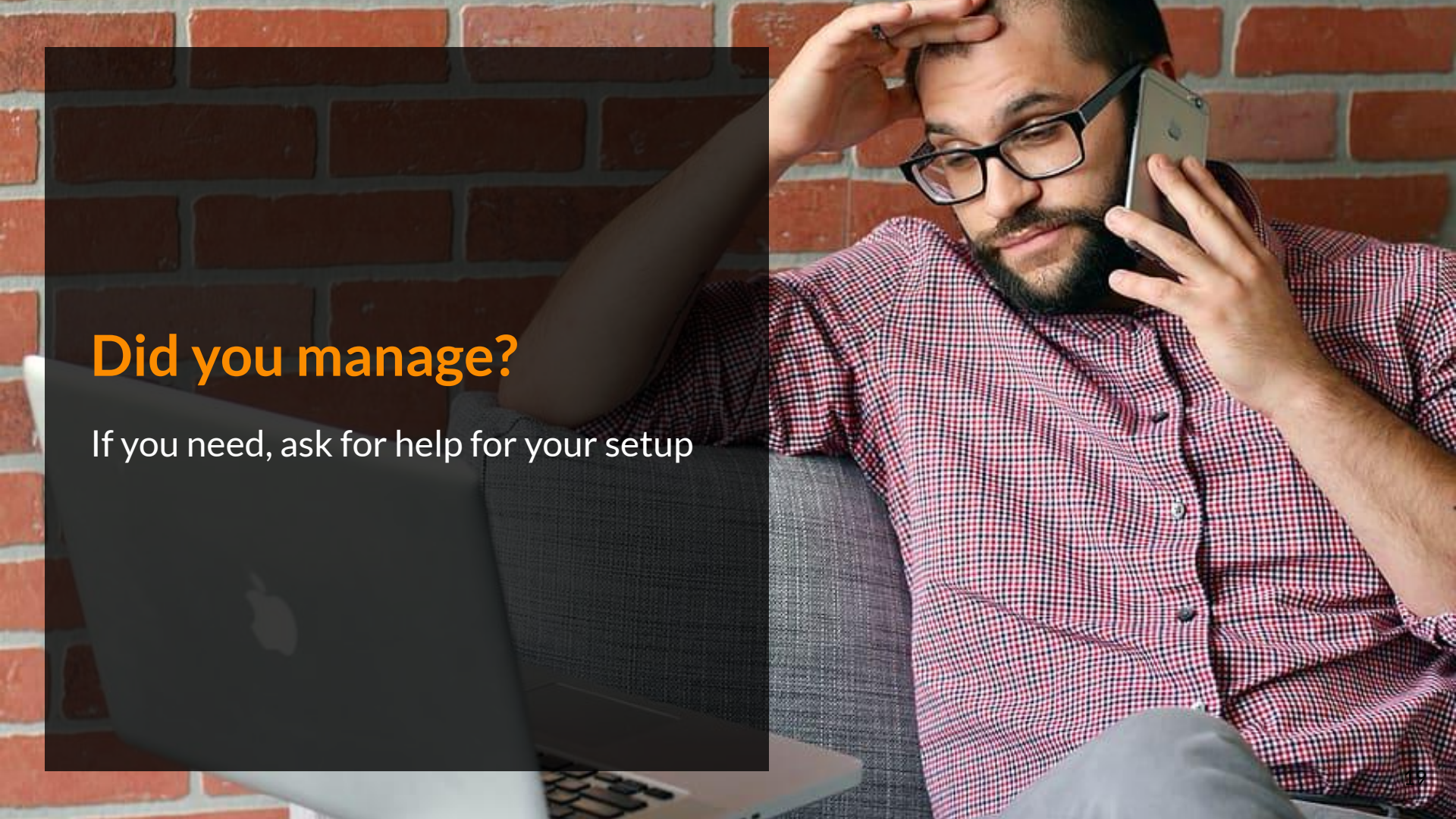
Example

Let's try and display Trento on the map.

With the containers all up, visit the frontend and press the "Trento" button. The coordinates should have changed, now hit "Submit".

Do you see Trento?

`localhost:8080`


Did you manage?
If you need, ask for help for your setup

Exercises

Map visualization

Embed categories in the event popups

Event filtering

Search for events of a specific category

Location searching

Get geographic coordinates by name through Nominatim

Exercise 1

NearMe Events

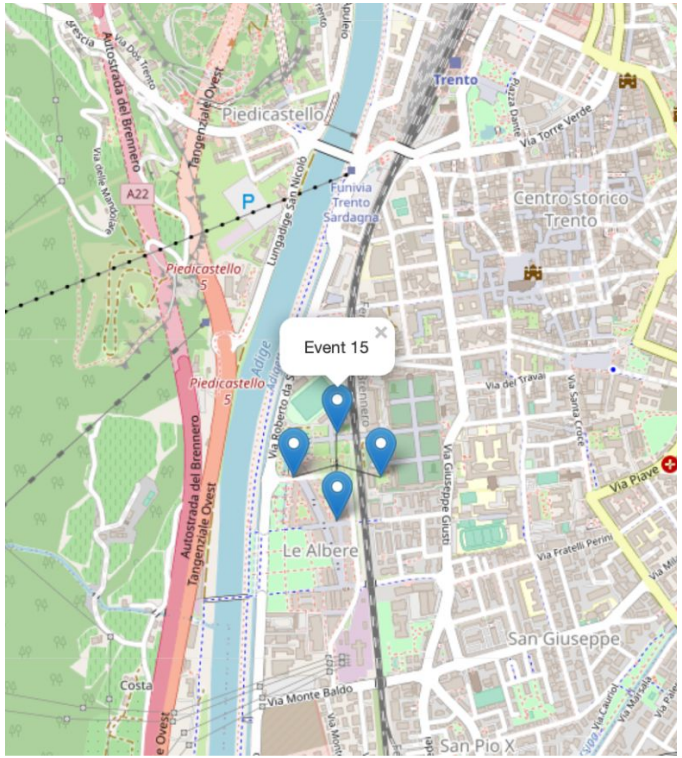
46.06787

Latitude

11.12108

Longitude

Submit



Event title

Exercise 1

NearMe Events

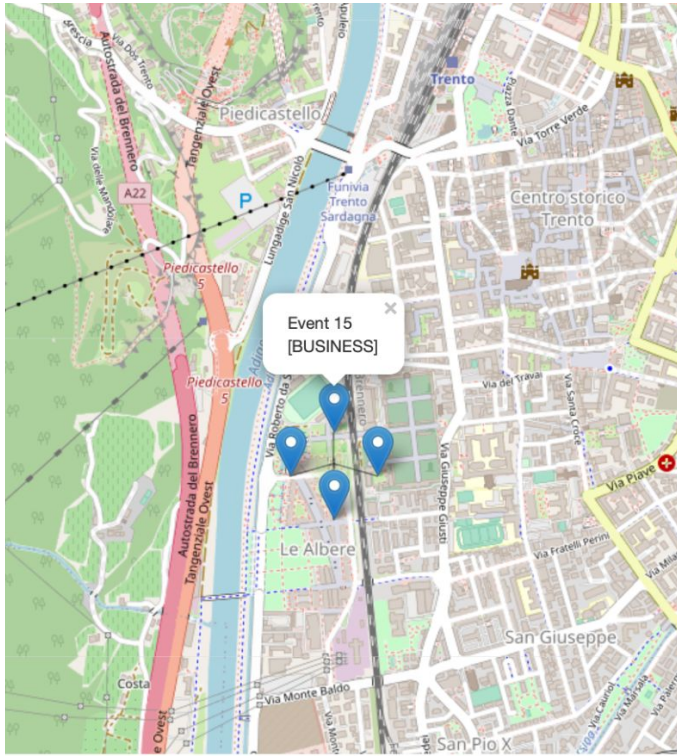
46.06787

Latitude

11.12108

Longitude

Submit



Event title + category

Exercise 1

Steps

1. Store the event category as a field of the json response fetched from REST Basics adapter for each event ID

2. Use the response to show category next to title on the event popup

Note that you have to restart the application once you have made the modifications.
Type **docker-compose up --force-recreate --build** to do that.

Exercise 1

Tip

REST Basics APIs

The variable **content** contains the fetched response from lab 4 client

lab4client.js

```
const prepareOneEvent = async (eventId) => {
    let url = `${process.env.LAB_4_URL}/v1/events/${eventId}`;
    const response = await fetch(url);
    const content = await response.json();
    return {
        id: eventId,
        title: content.title,
        lon: content.physicalAddress.geo.coordinates[0],
        lat: content.physicalAddress.geo.coordinates[1]
        /* Write your code here - exercise 1 */
    };
}
```

Exercise 1

SOLUTION

cat: content.category

lab4client.js

```
const prepareOneEvent = async (eventId) => {
    let url = `${process.env.LAB_4_URL}/v1/events/${eventId}`;
    const response = await fetch(url);
    const content = await response.json();
    return {
        id: eventId,
        title: content.title,
        lon: content.physicalAddress.geo.coordinates[0],
        lat: content.physicalAddress.geo.coordinates[1],
        cat: content.category
    };
}
```

Exercise 1

Tip

Leaflet quick start guide

The variable **events** contains the previously fetched response

```
// Set popups
if(events !== undefined){
  console.log("LENGHT: ", events.length)
  var markers = L.markerClusterGroup();
  for (var i = 0; i < events.length; i++) {
      // Create new marker
      marker = new L.marker([events[i].lat, events[i].lon])
      .bindPopup(events[i].title /* Write your code here - exercise 1 */);
      // Adding the marker into a Layer
      markers.addLayer(marker);
  }
  // Adding the layer into the map
  map.addLayer(markers);
```

webmap.js

Exercise 1

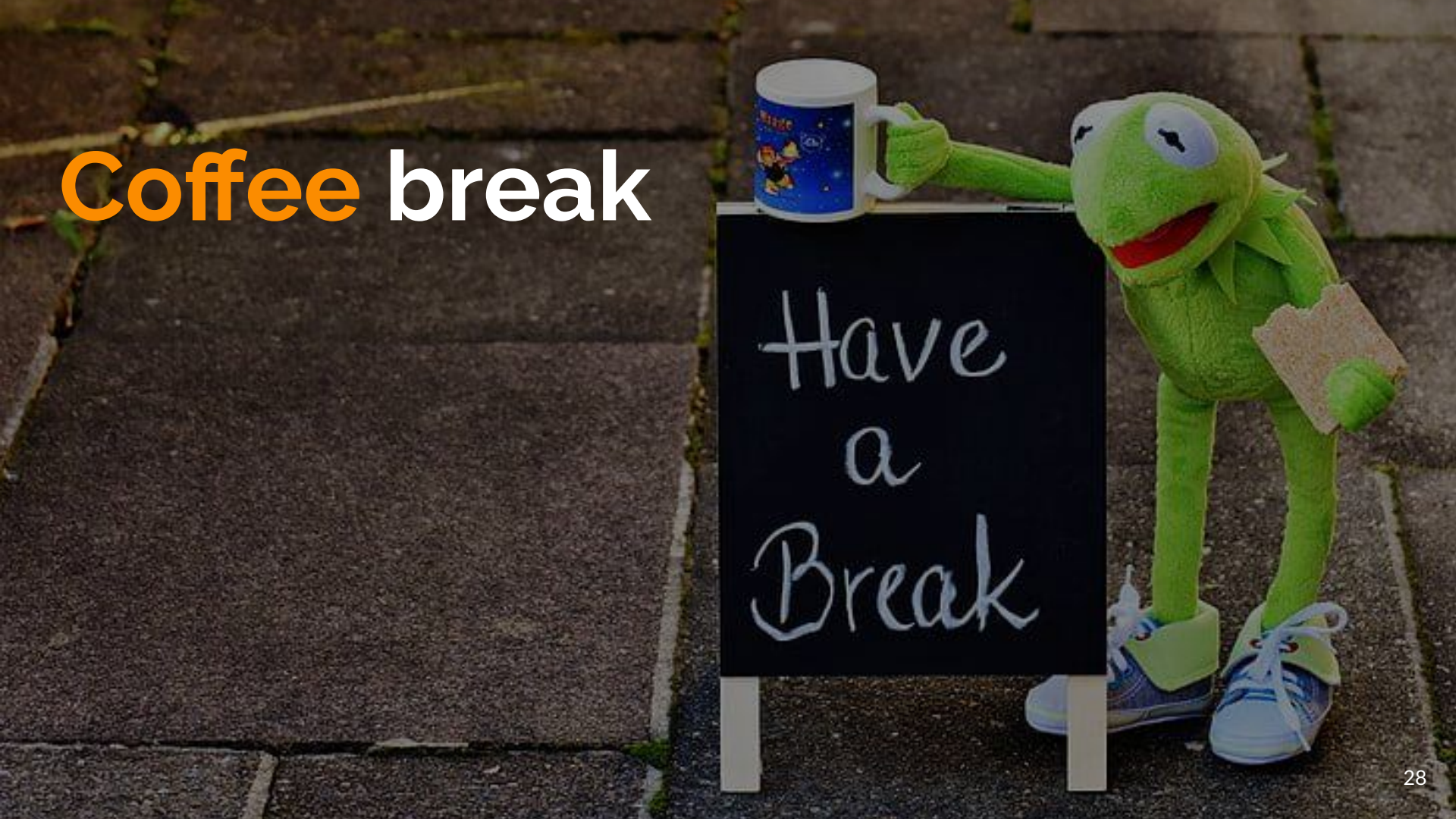
SOLUTION

+ "
[" + events[i].cat + "]"

```
// Set popups
if(events !== undefined){
  console.log("LENGHT: ", events.length)
  var markers = L.markerClusterGroup();
  for (var i = 0; i < events.length; i++) {
      // Create new marker
      marker = new L.marker([events[i].lat, events[i].lon])
      .bindPopup(events[i].title + "<br/>[" + events[i].cat + "]");
      // Adding the marker into a Layer
      markers.addLayer(marker);
  }
  // Adding the layer into the map
  map.addLayer(markers);
}
```

webmap.js

Coffee break



Exercise 2

45.454967

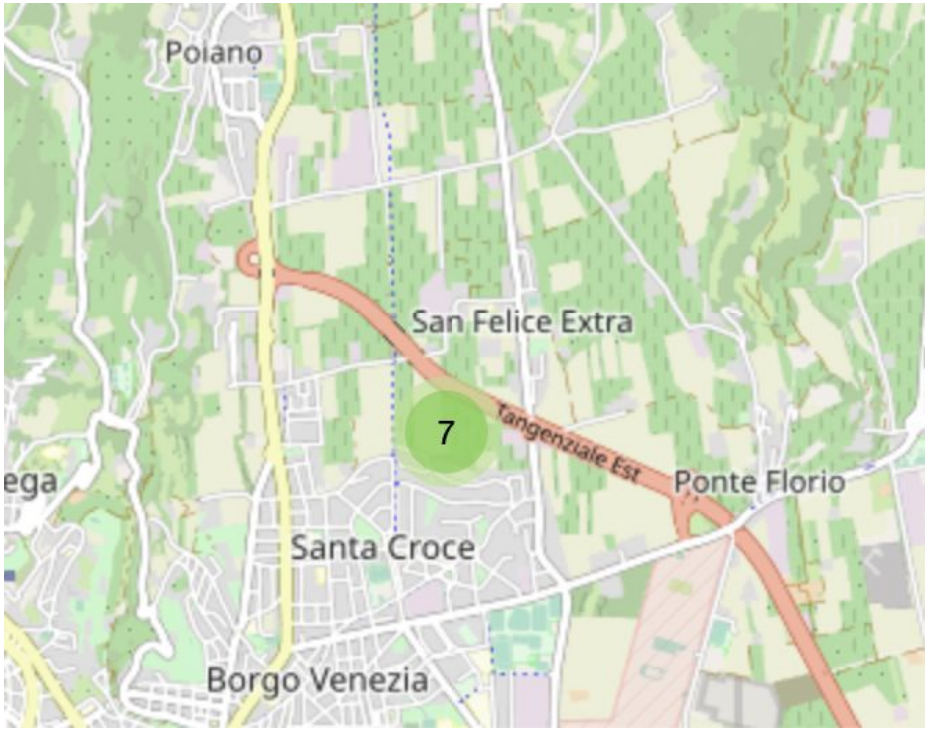
Latitude

11.029849

Longitude

Category

Submit



Exercise 2

45.454967

Latitude

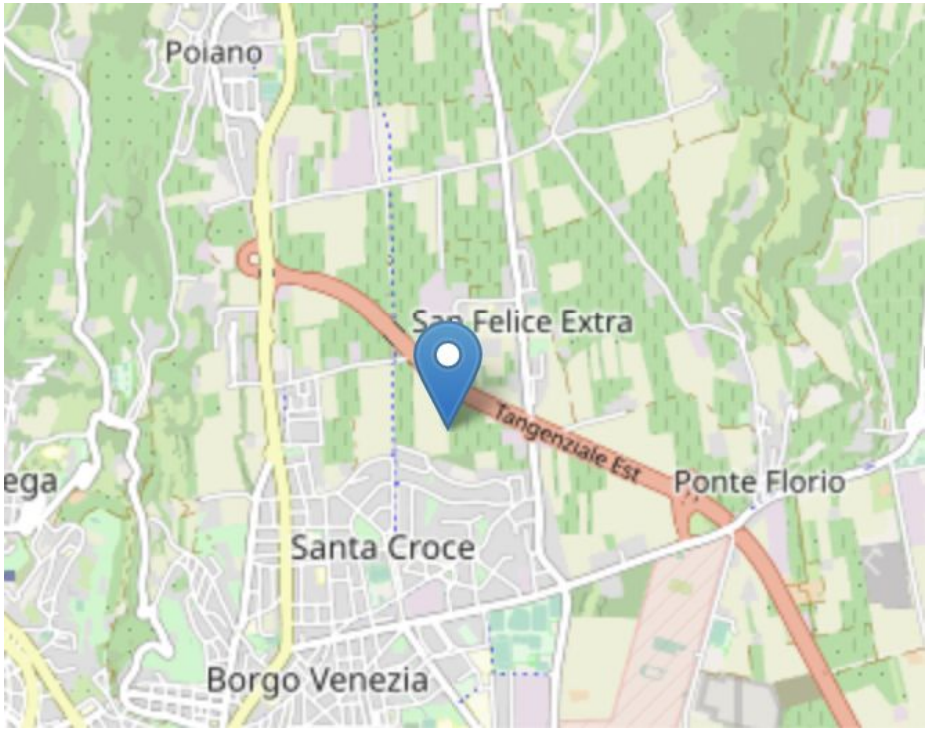
11.029849

Longitude

MEETING

Category

Submit



Exercise 2

What you should do:

1. Add a new text box to filter by category
2. Update the flow function to include category parameter

Hint:

Lab 4 service docs: https://restbasics.docs.apiary.io

Note that you have to restart the application once you have made the modifications.
Type **docker-compose up --force-recreate --build** to do that.

Exercise 2

index.html

```
<form class="w3-container" action="/flow" method="get">
  <p>
    <input class="w3-input" type="number" id="lat" name="lat" step="any" required min="0">
    <label for="lat">Latitude</label></p>
  <p>
    <input class="w3-input" type="number" id="lon" name="lon" step="any" required min="0">
    <label for="lon">Longitude</label></p>
  <!-- snip -->
</form>
```

Exercise 2

index.js

```
async function flow(lat, lon){
  // Read lab 4 data
  const eventIds = await lab4client.searchEvents(lat, lon, "");
  json_events = await lab4client.prepareEventsForMap(eventIds);
}

// GET flow function - given lat & lon, retrive JSON with events
router.get("/flow", async function (req, res, next) {
  // Read params
  let lat = req.query.lat;
  let lon = req.query.lon;

  // Query REST basic to get data
  await flow(lat, lon);

  // Redirect to standard route --> '/'
  await res.redirect('../?' + new URLSearchParams({lat:lat, lon:lon}));
});
```

Solution for Exercise 2

Skip the next two pages if you still want to work on the solution.

Exercise 2

index.html

```
<form class="w3-container" action="/flow" method="get">
  <p>
    <input class="w3-input" type="number" id="lat" name="lat" step="any" required min="0">
    <label for="lat">Latitude</label></p>
  <p>
    <input class="w3-input" type="number" id="lon" name="lon" step="any" required min="0">
    <label for="lon">Longitude</label></p>
  <p>
    <input class="w3-input" type="text" id="cat" name="cat">
    <label for="cat">Category</label></p>
  <!-- snip -->
</form>
```

Exercise 2

index.js

```
async function flow(lat, lon, cat){
  // Read lab 4 data
  const eventIds = await lab4client.searchEvents(lat, lon, cat);
  json_events = await lab4client.prepareEventsForMap(eventIds);
}

// GET flow function - given lat & lon, retrive JSON with events
router.get("/flow", async function (req, res, next) {
  // Read params
  let lat = req.query.lat;
  let lon = req.query.lon;
  let cat = req.query.cat;

  // Query REST basic to get data
  await flow(lat, lon, cat);

  // Redirect to standard route --> '/'
  await res.redirect('../?' + new URLSearchParams({lat:lat, lon:lon}));
});
```

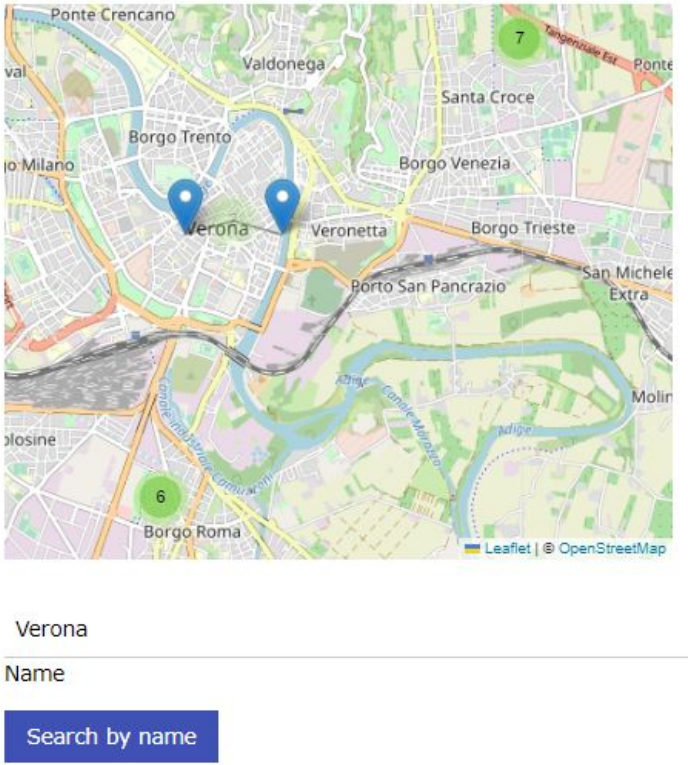
Exercise 3

Get geographic coordinates by name through Nominatim

Steps:

1. Complete the form into index.html page;

2. Fetch the request to nominatim.



Note that you have to restart the application once you have made the modifications.
Type **docker-compose up --force-recreate --build** to do that.

Exercise 3

Step one

Tip

Use the following **attributes** for the input text-box:

class = "w3-input"

type = "text"

id= "name"

name = "name"

required

index.html

```
<form class="w3-container" action="/searchOSM" method="get">
  <!-- Your code here / exercise 3 -->
</form>
```

Note that form action call "searchOSM" route.

This route is already handled in the index.js file.

Exercise 3

Step one - solution

```
<!-- Your code here / exercise 3 -->
<form class="w3-container" action="/searchOSM" method="get">
  <p>
    <input class="w3-input" type="text" id="name" name="name" required>
    <label for="name">Name</label>
  </p>
  <input id="sbname" type="submit" value="Search by name" class="w3-btn w3-indigo">
</form>
<br><br>
```

Exercise 3

Step two

See the Nominatim documentation here

index.js

```
// Set the API URL
// Your code here / exercise 3
let nominatim_api = "";

fetch(nominatim_api + new URLSearchParams({q:name, format:'json'})).then(async(response)=>{
```

Exercise 3

Step two - solution

```
// GET searchOSM function
router.get('/searchOSM', (req, res, next) => {
  // Read params
  let name = req.query.name;

  // Set the API URL
  // Your code here / exercise 3
  let nominatim_api = "https://nominatim.openstreetmap.org/search?";
```

Mini Assignment

Using the code written in the previous exercises, display the events present on the map given the name of a place.

Retrieve the events from Lab4's service as in the normal flow of operation to answer this question:

How many events are there in Milan?

Note that you have to restart the application once you have made the modifications.

Type **docker-compose up --force-recreate --build** to do that.

Tip

Call the flow function in the function written in Exercise 3 after the Nominatim call.

Good luck on your assignment!

We hope you have enjoyed this lab session!

Please submit your answers in the form you'll find on the Telegram group.


Thank you for your attention!