

# RailInspector



Programa de análisis de datos de sensores a tiempo real

## Características:

Escrito utilizando React, es un programa que hemos diseñado para mostrar datos a tiempo real de sensores de las vías de Euskotren. Estos datos se añaden a la base de datos, en este caso PostgreSQL y cogemos los datos para hacer varias cosas en la página. Con el uso de los datos, podemos utilizar un par de librerías: Leaflet y Recharts. Para el uso de datos hemos creado una API en node, a la cual podemos introducirles queries y sacar datos en Json para luego parsearlas y usarlas como se deben.

## PostgreSQL:

La base de datos es lo menos que he tenido que tocar, porque para eso hay una persona que trabaja en ello. Yo solo lo he tenido que crearla localmente, e importar los datos que me han pasado. Interpretarla y preguntar que queries quieren que haga. Poco a poco hemos ido haciéndolo todo, así que ningún problema. Este es uno de los ejemplos:

	id_trackgeometry [PK] integer	line character (10)	track character (10)	pk real	track_measure_date timestamp without time zone	 ubicacion geometry	 station boolean
1	1	HN-LO	VIA1	0.136	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
2	2	HN-LO	VIA1	0.13625	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
3	3	HN-LO	VIA1	0.1365	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
4	4	HN-LO	VIA1	0.13675	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
5	5	HN-LO	VIA1	0.137	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
6	6	HN-LO	VIA1	0.13725	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
7	7	HN-LO	VIA1	0.1375	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
8	8	HN-LO	VIA1	0.13775	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
9	9	HN-LO	VIA1	0.138	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
10	10	HN-LO	VIA1	0.13825	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
11	11	HN-LO	VIA1	0.1385	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
12	12	HN-LO	VIA1	0.13875	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
13	13	HN-LO	VIA1	0.139	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
14	14	HN-LO	VIA1	0.13925	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
15	15	HN-LO	VIA1	0.1395	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
16	16	HN-LO	VIA1	0.13975	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
17	17	HN-LO	VIA1	0.14	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
18	18	HN-LO	VIA1	0.14025	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
19	19	HN-LO	VIA1	0.1405	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
20	20	HN-LO	VIA1	0.14075	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
21	21	HN-LO	VIA1	0.141	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
22	22	HN-LO	VIA1	0.14125	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
23	23	HN-LO	VIA1	0.1415	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
24	24	HN-LO	VIA1	0.14175	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	false
25	25	HN-LO	VIA1	0.142	2019-09-20 10:02:45.847	0101000000AD7A3703D0AFFBF295C8FC2F5A845...	true

# API en node.js:

También he montado una API en node.js. Me ha ayudado Carlos, uno de mis compañeros en CEIT, por que yo solo sabia hacerlos en PHP y laravel, pero ha sido facilísimo porque eran similares. Hacer la conexión a la base de datos, hacer las rutas, en este caso dinámicas, que reciban datos en string para luego utilizar en las queries dinámicas. Todo lo más dinámico y adaptable posible. es una buena práctica para luego mantener e innovar en el código.

## Rutas:

```
JS router.js > ...
1  import router from "express";
2  import {getAlign, getAllCoordinates, getLevel} from "../Controllers/controller.js";
3  const Router = router();
4  Router.get("/Coordenadas", getAllCoordinates)
5  Router.get("/Niveles/:lineName", getLevel)
6  Router.get("/Alineacion/:lineName", getAlign)
7  export default Router;
```

## Controladores:

```
1  import db from "../Services/db.js";
2  const getAllCoordinates = (req, res) => {
3      // db.any('SELECT id_trackgeometry, line, ARRAY[ST_X(ubicacion), ST_Y(ubicacion)] as latlon FROM public.trackgeometry');
4      db.any('SELECT line, json_agg(ARRAY[ST_X(ubicacion), ST_Y(ubicacion)]) as latlon FROM public.trackgeometry');
5      res.send(data);
6  }
7
8  const getLevel = (req, res) => {
9      const lineName = req.params.lineName;
10     db.any('SELECT line, json_agg(json_build_array(pk, level_l_d1, level_r_d1)) AS data FROM public.trackgeometry WHERE line = $1');
11     res.send(data);
12 });
13
14
15 const getAlign = (req, res) => {
16     const lineName = req.params.lineName;
17     db.any('SELECT line, json_agg(json_build_array(pk, align_l_d1, align_r_d1)) AS data FROM public.trackgeometry WHERE line = $1');
18     res.send(data);
19 });
20
21
22 export{
23     getAllCoordinates,
24     getLevel,
25     getAlign,
26 }
```

## Conexiones:

```
1 import pgPromise from 'pg-promise';
2 import dotenv from "dotenv"
3 const pgp = pgPromise();
4 dotenv.config()
5 const db = pgp(`postgres://postgres:${process.env.DB_PASSWORD}@10.63.27.38:5432/railinspector`);
6 export default db;
7
```

## Index:

```
1 import express from "express";
2 import http from "http";
3 import Router from "./router.js";
4 const app = express()
5 app.use(express.json())
6 app.use(express.urlencoded({extended:true}))
7 app.use((req, res, next)=>{
8   res.header('Access-Control-Allow-Origin', "*");
9   res.header('Access-Control-Allow-Methods', "GET, PUT, POST, PATCH");
10  res.header('Access-Control-Allow-Headers', "Content-Type");
11  next();
12 })
13 app.use(Router)
14 app.get("/", (req, res)=>res.send("API levantada"))
15 http.createServer(app).listen(process.env.PORT || 3010)
16
```

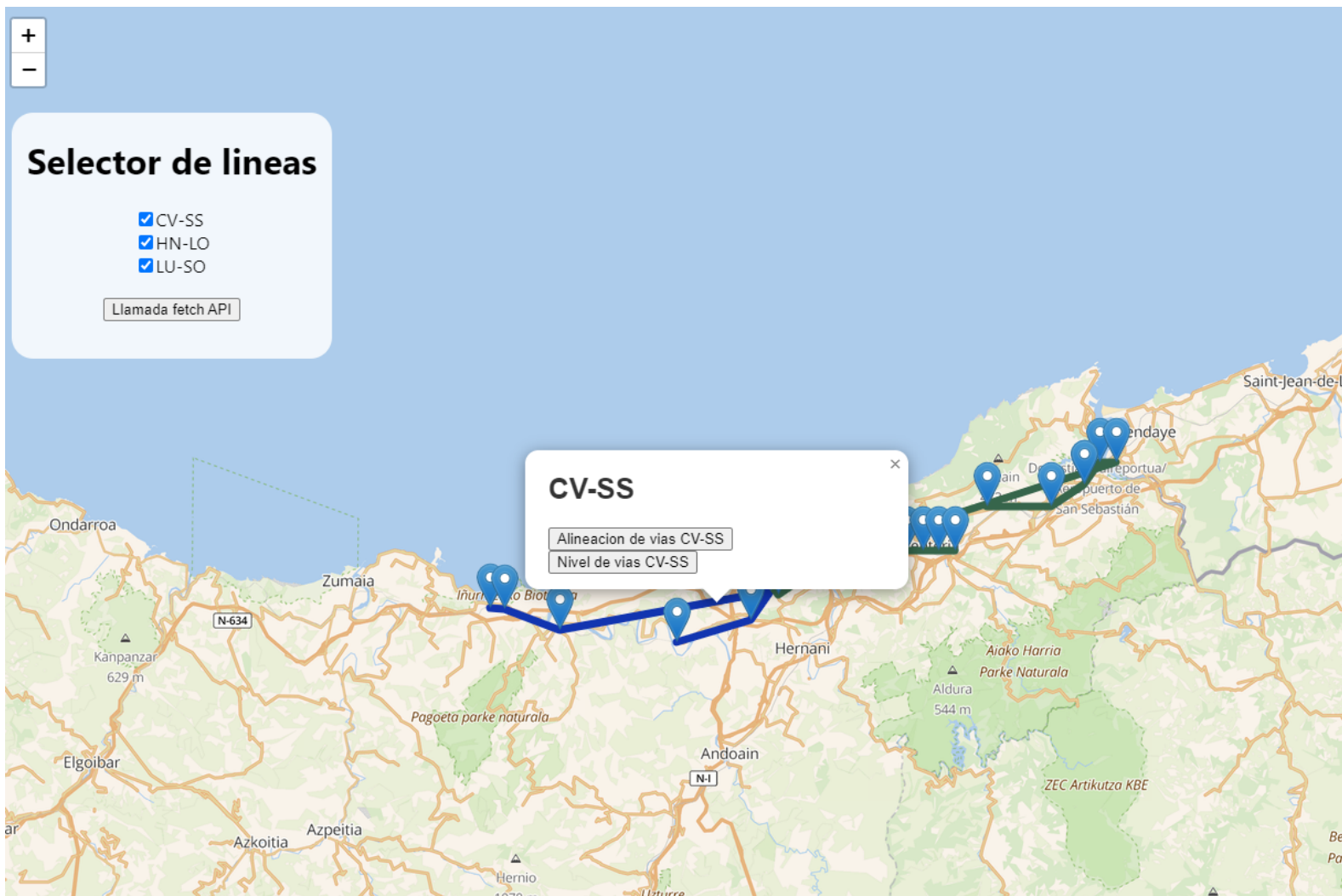
## Leaflet:

La base de datos contiene rutas, coordenadas, sensores de estaciones y de las vías en general. Haciendo una query podemos sacar solo las ubicaciones de las estaciones, y visualizar las rutas en el mapa. Es una manera más visual, interactiva e interesante de escoger los datos que queremos ver más tarde. También tenemos que mostrar y esconder las rutas, porque hay veces que se solapan unas encima de otras porque hacen el mismo camino. Lo haremos usando checkbox. Y hacer un pop up que se abra cuando clickeamos en las líneas con botones que usaremos luego. El proceso es el siguiente:

1. Escribir un fetch contra la ruta coordenadas (La API contiene una query específica para sacar justo los datos que queremos)
2. Recibir y parsear los datos del json
3. Adaptarlo para Leaflet

4. Hacer una variable y html para el checkbox
5. Conectarlo todo con un evento, un onClick que llama a una constante en este caso

Resultado:



## Recharts:

Para la visualización de datos, primero quisimos usar Grafana, por que nos hacia una API, con la conexión de la base de datos y todo, y los gráficos que podemos crear eran fáciles, vistosos, y aptos para una base de datos “big data” como la nuestra. Sin embargo, por culpa de problemas, principalmente de CORS, decidimos que lo mejor sería usar otra librería, y escogimos Recharts. La parte de la API es similar a la parte de leaflet. No solo por que es la misma API, sino que lo único que

cambia es el ruteo y las querys del controlador. Este es un ejemplo de ruta y query dinámicas:

```
Router.get("/Niveles/:lineName", getLevel)
```

```
const getLevel = (req, res) => {  
  const lineName = req.params.lineName;  
  db.any('SELECT line, json_agg(json_build_array(pk,  
level_l_d1, level_r_d1)) AS data FROM trackgeometry WHERE  
line = $1 GROUP BY line ORDER BY line  
ASC;', [lineName]).then((data)=>{  
    res.send(data);  
  });  
}
```

Al clicar en los botones del leaflet, pasamos el nombre de la línea en el fetch. Aquí se recibe, se coloca el dato en la query, y salen los datos de esa ruta que hemos seleccionado, y así para todas las rutas que tengamos. Ahorra mucho trabajo porque no tenemos que hacer querys uno por uno. Lo que nos devuelve es un Json de un tamaño considerable, ya que aquí queremos los datos de todos los sensores, no solo de las estaciones. Hay que parsear el Json, separarlo en el tipo de dato, nombrarlos y meterlos en un array para que luego recharts los lea. Lo de recharts es pan comido. Es juntarlo con el array, y llamar al tipo de dato que queramos que vaya en el eje X o Y.

El grafico:

```
<AreaChart  
  width={700}  
  height={400}  
  data={data} <=este es el array con los datos  
  margin={{  
    top: 10,  
    right: 40,
```

```

        left: 0,
        bottom: 0,
      }}
    >
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="pk" tick={{ dy: 2, dx: 22 }}
interval={Math.ceil(data.length / 10)} />
    <YAxis />
    <Tooltip />

```

Por predeterminado, si no se pone que en el eje X va un dato en concreto, recharts pondrá todo en el eje Y y eje x será la cantidad de datos que le hayamos pasado en número.

```

    {areaVisibility[0] && (
      <Area
        name="left"
        type="monotone"
        dataKey="left"
        stackId="1"
        stroke="#5ebfff"
        fill="#5ebfff"
      />
    )}

```

```

    {areaVisibility[1] && (
      <Area
        name="right"
        type="monotone"
        dataKey="right"
        stackId="1"

```

```

        stroke="#ff6857"
        fill="#ff6857"
    />
    )}
</AreaChart>

```

Esto:

```
{areaVisibility[0] && (
```

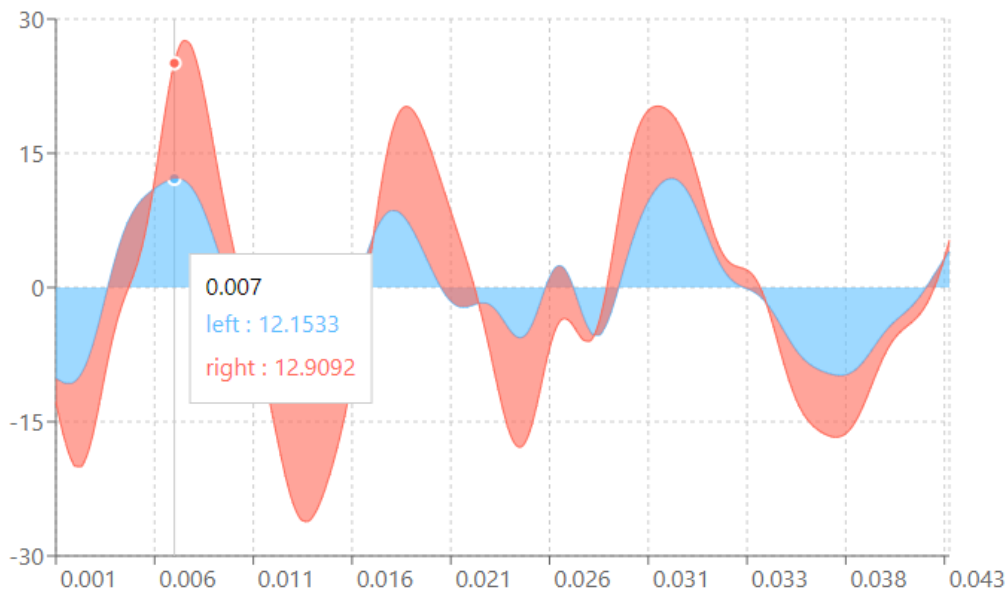
Es para juntarlo con el checkbox. Si es true, se mostrará, si es false no.

El resultado:

Mostrando Alineacion de la ruta LU-SO

☒ Mostrar area izquierda

☒ Mostrar area derecha



Mostrando Niveles de la ruta LU-SO

☐ Mostrar area izquierda

☒ Mostrar area derecha

