

BASES DE DATOS AVANZADAS

PRÁCTICA 1: GIS

Juan Casado Ballesteros
Gino Cocolo Rodríguez



Universidad
de Alcalá

ÍNDICE

INTRODUCCIÓN3

ÁREAS FIR/UIR3

ÁREAS TMA3

RUTAS DE LOS AVIONES3

MANEJO DE DATOS ESPACIALES3

VISUALIZACIÓN DE DATOS ESPACIALES4

TRABAJO EXTRA4

DISEÑO DE LA BASE DE DATOS5

TABLAS Y COLUMNAS5

INSERCIÓN DE DATOS6

CÁLCULOS REALIZADOS EN LA BASE DE DATOS7

ACCESO A LA BASE DE DATOS9

REALIZACIÓN DE LAS CONSULTAS9

INTERFAZ REST9

SERVIDOR10

DISEÑO DE LA INTERFAZ WEB12

OPEN LAYERS12

REACT12

CONEXIÓN CON LA BASE DE DATOS13

ACCESO A LA INTERFAZ13

OTRAS FORMAS DE VISUALIZAR LOS DATOS17

QGIS17

DESPLIEGUE DE LA APLICACIÓN18

DOCKER18

INTRODUCCIÓN

Nuestro objetivo en esta práctica es el de crear una aplicación capaz de calcular el coste de las rutas de los aviones en función de la trayectoria recorrida.

Para lograrlo deberemos manejar los siguientes tipos de datos de datos:

Áreas FIR/UIR

Cubren diversas regiones de la Península y de Canarias. Todo avión dentro de estas áreas deberá pagar en función de la siguiente ecuación. Un avión solo puede ser cobrado por estar en una de estas áreas a la vez. Cabe destacar que los aviones pagarán más cuanta más distancia recorran en estas áreas.

$$\text{PRECIO} = [51,08 \text{ si Península o } 43,73 \text{ si Canarias}] * \text{KM}/100 * (\text{TONELADAS_METRICAS}/50)^0,5$$

Áreas TMA

Cubren regiones próximas a los aeropuertos. Estas áreas son en realidad volúmenes con altura máxima en los 20.000 pies. Un avión solo puede ser cobrado por estar en una de estas áreas a la vez, si además está en un área FIR/UIR será cobrado como si solo estuviera en TMA.

Cabe destacar que no importa la distancia que se recorra dentro de las áreas TMA.

$$\text{PRECIO} = [18,72 \text{ o } 16,84 \text{ o } 14,04 \text{ según aeropuerto}] * (\text{TONELADAS_METRICAS}/50)^0,7$$

Rutas de los aviones

Nos importa la ruta que han recorrido, en concreto la lista de puntos por los que han pasado para poder calcular dentro de qué área estaban, así como la distancia recorrida dentro de cada una de ellas. Adicionalmente nos importa también la altura en las que estaban en cada uno de los puntos.

También necesitaremos conocer el avión que realizó la ruta para poder saber sus MTWO o peso máximo autorizado en el despegue en toneladas métricas.

Manejo de datos espaciales

Como podemos ver, muchos de los datos que manejaríamos tienen referencias espaciales, en concreto los puntos que definen las áreas y los puntos que definen la trayectoria de los aviones.

Utilizaremos por tanto una base de datos capaz de manejar datos con referencias espaciales para que nos ayude a almacenar los datos, así como a obtener las relaciones entre ellos. En nuestro caso utilizaremos la base de datos PostgreSQL con su extensión GIS postgis.

La ventaja de utilizar un sistema GIS recae en tres pilares fundamentales:

- **Estandarización:** según hemos estado trabajando con la base de datos y la hemos ido integrando con otras partes de nuestra aplicación hemos comprobado la sencillez de hacer esto debido a que todos los componentes eran capaces de entender los formatos en que estaban expresadas las referencias espaciales.
- **Adaptación:** Una herramienta GIS está ya preparada para trabajar con referencias espaciales de forma óptima sin que debamos de hacer ninguna configuración adicional sobre ella. En la documentación de postgis hemos encontrado que una técnica recurrente que utilizan son los árboles quad-tree para indexar los datos que contiene referencias espaciales. Este tipo de índices utilizan nodos con cuatro hijos que subdividen regiones de espacio un cuarto menor que las que su padre cubría.

También es digno de ser mencionado lo sencillo y cómodo que resulta almacenar los datos espaciales en postgis utilizando el tipo geom para las columnas de las tablas.

- **Reutilización:** la mayoría de las operaciones que hemos deseado realizar sobre los datos espaciales estaban ya implementadas en postgis. Adicionalmente explorando la documentación hemos encontrado otras operaciones que, aunque no nos han hecho falta para completar la práctica nos han parecido de gran utilidad.

Visualización de datos espaciales

Adicionalmente desarrollaremos una aplicación web que nos permita manejar y visualizar los datos desde una interfaz cómoda. Dicha aplicación consistirá en una pequeña interfaz creada con React sobre la que incorporamos las librerías de Open Layers y Turf.js para visualizar los datos que provienen de la base de datos sobre un mapa.

Los mapas que utilizamos provienen de Open Street Maps, un proveedor de mapas gratuito cuyos mapas son editables por cualquier persona. Explorando este proveedor de información hemos descubierto que no solo proporciona mapas si no que también proporciona otros muchos tipos de datos como edificios, puntos de interés en las ciudades o carreteras.

Trabajo extra

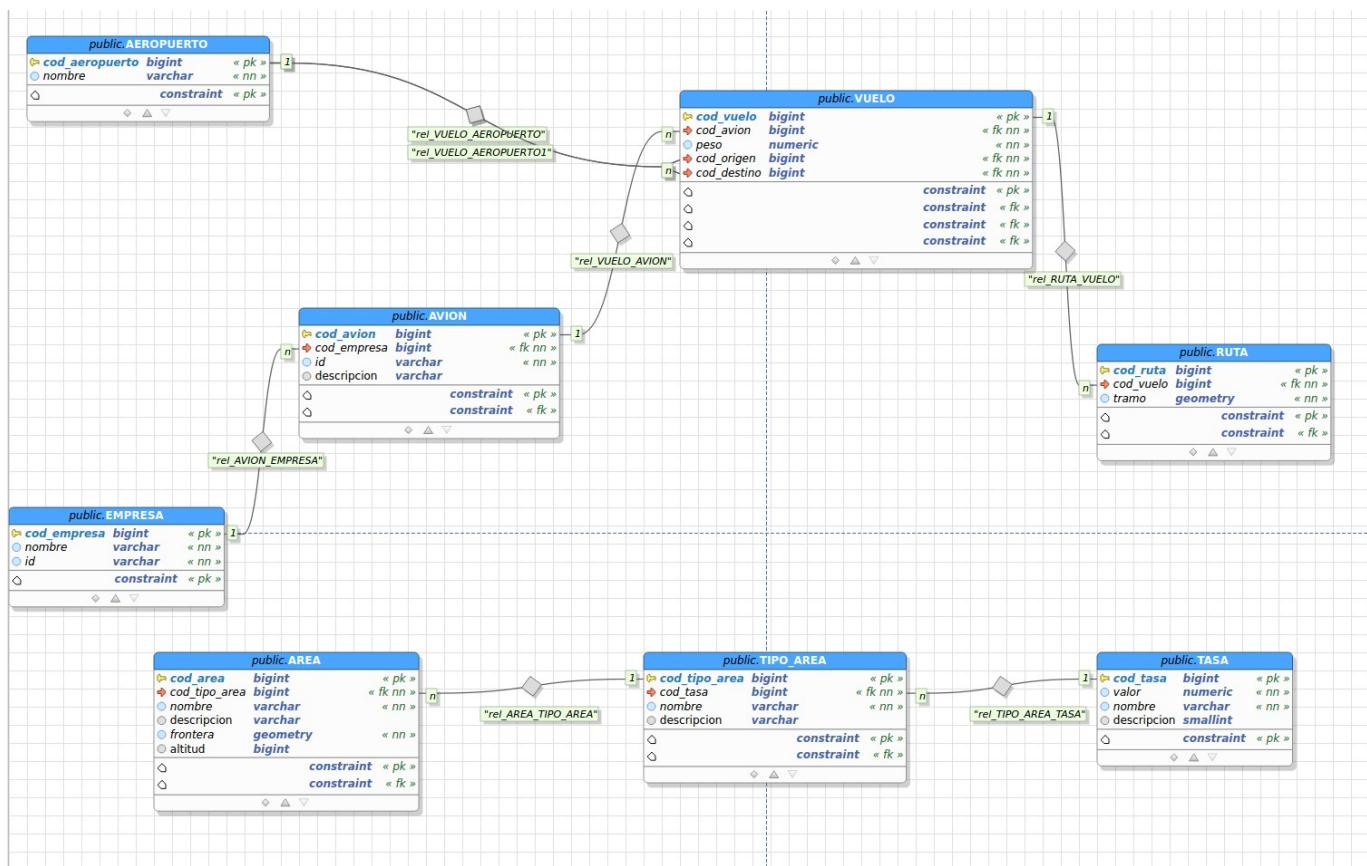
- Hemos probado otras herramientas GIS, en concreto QGIS, comprobando que se integraban adecuadamente con nuestra base de datos.
- Todas las partes de la aplicación han sido desarrolladas dentro de contenedores Docker. Esto nos ha permitido trabajar sin tener problemas de instalación de las distintas herramientas y sin tener problemas de versiones o dependencias.
- Hemos desarrollado un servidor web que crea una interfaz REST frente a la base de datos. Dicho servidor es un Tomcat que sirve una aplicación JAVA que utiliza Spring para crear la interfaz REST. El uso de este servidor nos ha facilitado en gran medida el trabajo pues es un intermediario que simplifica el acceso a los datos almacenados en la base de datos desde la interfaz web.

DISEÑO DE LA BASE DE DATOS

El primer paso para completar la práctica fue crear la base de datos. Utilizaremos PostgreSQL, que junto a su extensión postgis será capaz de manejar los datos espaciales que necesitamos utilizar.

Tablas y columnas

Para poder crear la base de datos primero tuvimos que diseñar su estructura lo cual pudimos hacer mediante el siguiente diagrama de tablas tras haber inspeccionado el formato de los datos que deberíamos almacenar en ella.



- **TASA**: Contiene el valor fijo a aplicar y un nombre para identificarlo.
- **TIPO_AREA**: Define los distintos tipos de área existentes (FIR/UIR Península o los grupos de TMAs) y se relacionan con una tasa concreta.
- **AREA**: Todas las distintas áreas en el sistema, cada una perteneciente a un tipo, aquí se define mediante una *geometry* sus límites, un *Polygon*, y en caso de ser aplicable, la altitud hasta la que surten efecto, así como un nombre para identificarlas.
- **EMPRESA**: Simplemente empresas dadas de alta y su nombre e identificativo.
- **AVIÓN**: Contiene los datos básicos de los aviones y la empresa a la que pertenecen.

- **AEROPUERTO:** Nombres de aeropuertos.
- **VUELO:** Alberga los datos referentes a un vuelo realizado por un avión, como el aeropuerto de origen y destino y el peso con el que ha despegado.
- **RUTA:** Los distintos tramos que ha ido recorriendo el avión durante el vuelo y la altitud en la que lo ha hecho. Estos tramos están definidos como una *geometry*, una *LineString*.

Se han mantenido los datos necesarios para la demostración que nos atañe, pero al igual que el servidor, está planteado para un desarrollo más extensivo, pudiendo llegar a tener información más detallada de aeropuertos, como su localización para representarlos en los mapas, poder desarrollar funciones de cálculos para todos los vuelos de un avión o de una empresa concreta y realizar una facturación completa en tramos para dicha empresa, incluir distintos tipos de áreas sin influir en el resto del modelo al estar tan desacoplado, etc.

Al igual que, simplemente insertando una nueva tabla *USUARIO* y relacionándolo con *EMPRESA*, se podría permitir visualizar los datos a trabajadores respectivos a su empresa y seguir ampliando funciones. Es una buena base de cara a seguir construyendo sobre ella sin tener que realizar cambios significativos.

Inserción de datos

Las geometrías de las áreas FIR/UIR y TMA las encontramos en el documento alojado en la siguiente dirección: <https://docplayer.es/docview/67/57005122/#file=/storage/67/57005122/57005122.pdf>

Las rutas de los aviones las obtuvimos de la siguiente página <https://es.flightradar24.com>

Los datos los modificamos manualmente para que se adapten al diseño de las tablas eliminando aquellos valores que no nos interesaban.

Se han insertado 4 vuelos distintos, las tres áreas FIR/UIR y un total de 10 áreas TMA.

Inserción de datos de las *geometries*: En el sistema se trabajan con dos *geometries*, los *Polygons* de las áreas y las *LineString* de las rutas. En el primer caso, a la hora de insertar, en el campo hay que introducir "*ST_GeometryFromText('Polygon((03.1000000 42.260000, 03.2339000 42.013200, ..., 03.1000000 42.260000))')*". Esto son las coordenadas de cada vértice del polígono que delimita la frontera del área especificada. A través de la función *ST_GeometryFromText* creamos una *geometry* empleando el formato de texto *WKT* (*Well-Known text*). De forma similar, a la hora de insertar el las rutas, utilizamos "*ST_MakeLine(ST_GeomFromText('POINT(-0.524100 51.477200)'), ST_GeomFromText('POINT(-0.546800 51.474600)'))*". Primero, con la misma función que para el polígono, definimos los dos puntos necesarios para el inicio y fin de la línea, y con esas dos *geometries* y la función *ST_MakeLine* ya lo tenemos listo.

Cálculos realizados en la base de datos

A la hora de trabajar, hay tres consultas relacionadas con *postgis* que merecen la pena destacar:

Selección de los datos de las *geometries*: A la hora de servir los datos de las áreas o las rutas, debemos enviar las coordenadas para poder representarlas en el mapa. Para ello, existe una función de *postgis* muy útil: “*ST_AsGeoJSON*”. Esto te devuelve al pasarle una *geometry* una cadena en formato JSON con el tipo de geometría que es y sus coordenadas. Como lo único que nos interesa son las coordenadas, a través de una función de PostgreSQL extraemos ese atributo en concreto, quedando de esta manera “*json_extract_path(ST_AsGeoJSON(frontera)::json, 'coordinates')*”. Con esto conseguimos una cadena de texto con solamente las coordenadas, listas para mandar. En el caso de que nos pidan toda la ruta realizada en un vuelo, lo que nos interesa es la unión de todos los tramos. Convenientemente, la función con la que habíamos insertado los datos de la ruta, *ST_MakeLine*, funciona también como una función de agregación al recibir un set de resultados. Así que de la misma forma que antes, extraemos las coordenadas resultantes de agregar todos los puntos previamente ordenados para poder dibujar la ruta en el mapa.

Cálculo del recorrido del vuelo en las distintas áreas: Cuando queremos calcular cuánto ha recorrido el avión por las distintas zonas, y cuál es el área vigente para el cobro, empleamos la siguiente consulta:

```
SELECT a.cod_area AS a_cod_area, a.nombre AS a_nombre, ta.cod_tipo_area AS  
a_ta_cod_tipo_area, ta.nombre AS a_ta_nombre, t.valor AS a_ta_t_valor, t.cod_tasa AS  
a_ta_t_cod_tasa,  
       sum(st_length(st_intersection(r.tramo, a.frontera)::geography)) / 1000 AS  
recorrido  
FROM "RUTA" r, "AREA" a  
LEFT JOIN "TIPO_AREA" ta USING(cod_tipo_area)  
LEFT JOIN "TASA" t USING(cod_tasa)  
WHERE r.cod_vuelo = #{codVuelo} AND st_intersects(r.tramo, a.frontera) AND ((t.cod_tasa > 2  
AND r.altitud < a.altitud) OR t.cod_tasa < 3)  
GROUP BY cod_area, a.nombre, ta.cod_tipo_area, t.valor, t.cod_tasa
```

La consulta se queda con aquellos tramos de la ruta de un vuelo concreto (*WHERE r.cod_vuelo = #{codVuelo}*) que se encuentran contenidos en un área, gracias a la función *st_intersects*, que recibiendo dos geometrías devuelve true si las dos comparten un espacio en común (*AND st_intersects(r.tramo, a.frontera)*), y discierne según la altura si debe ser el área de aproximación la que se debe de aplicar o no (*AND ((t.cod_tasa > 2 AND r.altitud < a.altitud) OR t.cod_tasa < 3)*). En esta consulta, el resultado es una sola línea por cada área por la que se ha pasado y un recorrido total en dicha área, de ahí la cláusula del *GROUP BY*. El cálculo de ese recorrido lo conseguimos gracias a la combinación de las funciones *st_intersection*, la cual al recibir dos geometrías devuelve otra con el espacio que tienen en común, y *st_length*, que calcula la longitud de esa geometría devuelta. Como la unidad que nos interesa es kilómetros, una manera de transformarlo es transformar la *geometry* a *geography* antes de pasarselo a

`st_length`, y eso ya nos lo devuelve en metros. Solo quedaría dividirlo entre 1.000, y como estamos calculando un total, pasarlo a la función de agregación `sum`, ya obteniendo el recorrido total en kilómetros en un área tras agrupar (`sum(st_length(st_intersection(r.tramo, a.frontera)::geography)) / 1000`).

Mostramos a continuación capturas realizadas sobre una instancia de pgAdmin conectada a nuestra base de datos.

The screenshot shows the pgAdmin interface with three main tabs: Query Editor, Scratch Pad, and Data Output. The Data Output tab is active, displaying a table of results from a query that joins areas with their corresponding tramo lengths and calculates a sum based on specific conditions.

```

1 select a.cod_area, a.nombre, ta.cod_tipo_area, ta.nombre, t.valor,
2      sum(st_length(st_intersection(r.tramo, a.frontera)))
3  FROM "RUTA" r, "AREA" a
4  LEFT JOIN "TIPO_AREA" ta USING(cod_tipo_area)
5  LEFT JOIN "TASA" t USING(cod_tasa)
6 WHERE st_intersects(r.tramo, a.frontera) AND ((t.cod_tasa > 2 AND r.altitud < a.altitud) OR t.cod_tasa < 3)
7 GROUP BY cod_area, a.nombre, ta.cod_tipo_area, t.valor
  
```

	cod_area	nombre	cod_tipo_area	nombre	valor	sum
1	1 Barcelona TMA		3 TMA G1		18.72	1.8383551018038857
2	2 Barcelona FIR/UIR		1 FIR/UIR Peninsula		51.08	4.355640944711582
3	4 Valencia TMA		3 TMA G1		18.72	1.6787056699309921
4	5 Canarias FIR/UIR		2 FIR/UIR Canarias		43.73	3.610210278532845
5	6 Canarias TMA		3 TMA G1		18.72	3.5968620612297664
6	7 Madrid FIR/UIR		1 FIR/UIR Peninsula		51.08	42.36545419981094
7	8 Madrid TMA		3 TMA G1		18.72	24.459127667449465
8	10 Galicia TMA		4 TMA G2		16.84	1.1147814513022898
9	11 Santander TMA		5 TMA G3		14.04	0.8999349282354103
10	12 Sevilla TMA		3 TMA G1		18.72	2.38505784873432

ACCESO A LA BASE DE DATOS

Hemos decidido crear un servidor que se situara entre la base de datos y los clientes web. Mediante el uso de dicho servidor los clientes podrán evitar realizar consultas directas a la base de datos de modo que podrán permitirse desconocer la estructura interna de esta. Tan solo conocerán una versión simplificada de la misma en la que las acciones que pueden realizar sobre ella se representan mediante una API REST. Dicho servidor ha sido desarrollado utilizando el framework Spring y la librería MyBatis principalmente. Dicha librería ofrece una interfaz simplificada sobre la librería JDBC que es la que realmente crea la conexión con la base de datos.

Realización de las consultas

Mybatis proporciona una solución a la hora de conectarse a una base de datos y mapear consultas con objetos de Java. A través de interfaces, se definen una serie de métodos, que llevan asociados consultas en ficheros XML, y a la hora de trabajar, desde el código no nos preocupamos de realizar la conexión con la base de datos, ni de realizar las consultas y parsear posteriormente los resultados al objeto deseado. En dichos ficheros XML primeramente relacionamos una columna o un nombre del resultado con un atributo del objeto, pudiendo mapear desde los objetos básicos hasta objetos propios definidos por el desarrollador. Luego podemos definir todas las consultas, especificando el tipo de objeto que devuelve y que recibe, en caso de tener parámetros. Podemos definir parámetros para utilizar en la consulta, como puede ser una clave primaria para la cláusula del *WHERE* o incluso un objeto, pudiendo acceder a los atributos de este y utilizarlos en la consulta.

src/main/resources:

- *mybatis*: Contiene el fichero `jdbc.properties` donde se configura el acceso a la base de datos, definiendo la dirección donde se encuentra y las credenciales para el acceso.
- *mybatis.mappers.**: Aquí es donde se encuentran todos los ficheros XML con los mapeos correspondientes y las consultas, todo métodos definidos en la interfaz que se encuentra en el paquete `org.uah.mmaa.features` con el que comparta nombre.

Interfaz REST

Spring es una cómoda solución para crear una API Rest, pudiendo definir fácilmente las rutas y un diseño Vista-Controlador. Además, con Spring Security ofrece la posibilidad de implementar niveles de seguridad y controles de acceso a cada ruta, pudiendo identificar al usuario y sus niveles de permisos. Permite al desarrollador centrarse en lo importante y dejar en manos de Spring la conexión y el manejo de la comunicación Cliente-Servidor.

src/main/java:

- *org.uah.mmaa.config.**: Se encuentran las diferentes configuración de MyBatis. Esto nos permite automatizar la conexión a base de datos, la forma de manejar las peticiones y el cómo

respondemos a estas peticiones. Un ejemplo de ello es serializar y deserializar automáticamente, utilizando la librería Gson, los objetos Java a formato JSON para crear las respuestas.

- *org.uah.mmaa.core.**: Formado por el conjunto de configuraciones relativas a los logs, excepciones, útiles, etc. Aquí es donde se inicializa el servidor, especificando las distintas configuraciones que tiene que seguir, así como la manera en la que debe escribir el log y una serie de respuestas a dar según el error que se pueda producir durante la ejecución.
- *org.uah.mmaa.features.**: Se encuentran las diferentes funcionalidad que aporta el sistema como son lo referente a los aviones,vuelos y rutas. Cada paquete aquí contiene todo lo relacionado con su módulo, desde los objetos Java (*Area*, *TipoArea*,...) hasta el Controller correspondiente encargado de mapear las distintas direcciones e invocar a los métodos correspondientes, como la interfaz para trabajar con la base de datos.

Finalmente, la interfaz REST expuesta consta de las siguientes rutas:

- **/**: muestra una página web de prueba para que podamos saber si el servidor está desplegado correctamente.
- **/area**: Devuelve un listado con todas las áreas, su tipo y la tasa que se cobrará por pasar por ella.
- **/avion/vuelos**: Proporciona un listado con todos los vuelos existentes indicando el avión que lo realizó y el lugar de origen y destino
- **/avion/vuelos/ruta?codVuelo=id**: Proporciona la ruta recorrida por el vuelo cuya id se haya indicado.
- **/avion/calcular?codVuelo=id**: Proporciona la factura correspondiente vuelo cuya id se haya indicado.

Servidor

Para desplegar la aplicación creada se utiliza el servidor Tomcat. Dicho servidor utiliza archivos WAR, que ubicados en el path indicado en la configuración crearán las rutas necesarias para derivar el tráfico entrante hacia ellos.

El servidor es muy cómodo de usar ya que puede ser desplegado en un contenedor sobre el que se monta un volumen. Todos los archivos WAR que se añadan al volumen serán lanzados por el servidor de forma automática lo que permite trabajar con este servidor de forma rápida y sencilla.

Mediante Graddle, que es el compilador utilizado, podemos generar automáticamente estos archivos WAR sobreescribiendo los antiguos para lanzar una nueva versión del servidor. Esta compilación se puede hacer también manualmente con jar -cvf.

Mostramos a continuación capturas mostrando los resultados de algunas de las peticiones que se pueden hacer sobre el servidor.

The screenshots illustrate the results of three API requests:

- Request 1: GET http://localhost:8080/sistema-modelos-avanzados/area**
- Request 2: GET http://localhost:8080/sistema-modelos-avanzados/avion**
- Request 3: GET http://localhost:8080/sistema-modelos-avanzados/avion/calculo?codVuelo=1**

Request 1: GET http://localhost:8080/sistema-modelos-avanzados/area

URL Parameter: codArea = 1

Response (JSON):

```

{
  "Root": {
    "Index 0": [
      {
        "codArea": "1",
        "nombre": "Barcelona TMA",
        "frontera": "[[31.42,26],[3.2339,42.0132],[3.2321,41.5633],[3.2321,41.5633],[31.42,26]]",
        "tipoArea": "TMA G1",
        "codTasa": "3",
        "nombre": "Tasa Aproximación G1",
        "valor": 18.72
      }
    ],
    "Index 1": [],
    "Index 2": [],
    "Index 3": [],
    "Index 4": [],
    "Index 5": [],
    "Index 6": [],
    "Index 7": [],
    "Index 8": [],
    "Index 9": []
  }
}

```

Request 2: GET http://localhost:8080/sistema-modelos-avanzados/avion

URL Parameter: codAvion = 1

Response (JSON):

```

{
  "Root": {
    "Index 0": [
      {
        "codAvion": "A321",
        "id": 1,
        "nombre": "Airbus A321"
      }
    ],
    "Index 1": [
      {
        "codAvion": "737-300",
        "id": 2,
        "nombre": "Boeing 737-300"
      }
    ],
    "Index 2": [
      {
        "codAvion": "A320",
        "id": 3,
        "nombre": "Airbus A320"
      }
    ]
  }
}

```

Request 3: GET http://localhost:8080/sistema-modelos-avanzados/avion/calculo?codVuelo=1

URL Parameter: codVuelo = 1

Response (JSON):

```

{
  "Root": {
    "conceptos": [
      {
        "Index 0": [
          {
            "area": [
              {
                "Index 0": [
                  {
                    "codArea": "7",
                    "nombre": "Madrid FIR/UR"
                  }
                ],
                "Index 1": [
                  {
                    "codTasa": "1",
                    "nombre": "FIR/UR Península"
                  }
                ],
                "Index 2": [
                  {
                    "codTasa": "1",
                    "nombre": "FIR/UR Península"
                  }
                ]
              }
            ],
            "Index 1": [
              {
                "tasa": [
                  {
                    "Index 0": [
                      {
                        "codTasa": "1",
                        "valor": 51.08
                      }
                    ],
                    "Index 1": [
                      {
                        "recorrido": [
                          {
                            "Index 0": [
                              {
                                "costeCalculado": "535.2152052769145"
                              }
                            ],
                            "Index 1": [
                              {
                                "costeCalculado": "273.3879268554479"
                              }
                            ],
                            "Index 2": [
                              {
                                "total": "6665.3327541160215"
                              }
                            ]
                          }
                        ],
                        "Index 3": [
                          {
                            "total": "6665.3327541160215"
                          }
                        ]
                      }
                    ],
                    "Index 4": [
                      {
                        "total": "6665.3327541160215"
                      }
                    ]
                  }
                ],
                "Index 5": [
                  {
                    "total": "6665.3327541160215"
                  }
                ]
              }
            ],
            "Index 6": [
              {
                "total": "6665.3327541160215"
              }
            ]
          }
        ],
        "Index 1": [
          {
            "Index 0": [
              {
                "total": "6665.3327541160215"
              }
            ],
            "Index 1": [
              {
                "total": "6665.3327541160215"
              }
            ],
            "Index 2": [
              {
                "total": "6665.3327541160215"
              }
            ]
          }
        ],
        "Index 3": [
          {
            "Index 0": [
              {
                "total": "6665.3327541160215"
              }
            ],
            "Index 1": [
              {
                "total": "6665.3327541160215"
              }
            ],
            "Index 2": [
              {
                "total": "6665.3327541160215"
              }
            ]
          }
        ]
      }
    ],
    "Index 1": [
      {
        "Index 0": [
          {
            "total": "6665.3327541160215"
          }
        ],
        "Index 1": [
          {
            "total": "6665.3327541160215"
          }
        ],
        "Index 2": [
          {
            "total": "6665.3327541160215"
          }
        ]
      }
    ],
    "Index 3": [
      {
        "Index 0": [
          {
            "total": "6665.3327541160215"
          }
        ],
        "Index 1": [
          {
            "total": "6665.3327541160215"
          }
        ],
        "Index 2": [
          {
            "total": "6665.3327541160215"
          }
        ]
      }
    ]
  }
}

```

Vemos que los datos se proporcionan en formato JSON siendo agregados de múltiples tablas listos para ser utilizados por los clientes web.

DISEÑO DE LA INTERFAZ WEB

Para crear la interfaz web hemos decidido utilizar Open Layers. Ésta es una librería capaz de visualizar mapas en una página web, así como de manejar datos espaciales. Para tratar los datos espaciales hemos también instalado y probado Turf.js. No obstante, debido a que Open Layers proporciona por sí sola todas las características de las que requería nuestra aplicación, finalmente Turf.js no ha sido utilizada.

Hemos comprobado que las características que proporciona Open Layers, aunque excelentes para crear mapas web interactivos, necesitan de apoyo externo para poder crear una interfaz. Es por ello por lo que hemos decidido utilizar también React.

Open Layers

Open Layers nos ha permitido visualizar sobre un mapa obtenido de OSM los datos almacenados en la base de datos. Adicionalmente hemos profundizado algo más en el uso de esta librería permitiendo a los usuarios interactuar con los datos que se les mostraban.

Las acciones que pueden hacer son reducidas, no obstante, nos han permitido entender un poco mejor como funciona la librería por dentro. Dichas acciones consisten en proporcionar información a los usuarios sobre los elementos que seleccionan en el mapa, así como visualizar punto a punto las trayectorias de los aviones pudiendo desplazar estos con un slider.

Respecto a cómo utilizar Open Layers, el concepto principal es que cada elemento debe pertenecer a una capa (layer) pero a su vez cada elemento es una capa en sí mismo. De este modo se crea una estructura de capas en forma de árbol a través de la cual se propagan los estilos, las interacciones y la configuración de la capa. Podemos por ejemplo aplicar un estilo para que todos los elementos dentro de una capa sean de color rojo y posteriormente sobre una rama de árbol decir que a partir de allí sean de color azul. Esto permite crear estilos base que sean modificados según los elementos se hacen más y más concretos.

Frente a otras librerías similares como Leaflet hemos observado que Open Layers requiere de mayor trabajo por parte del usuario para poder crear y manejar las capas. Hay distintos tipos de capas como las vectoriales, las raster o las XYZ. Estas últimas son para visualizar datos en formato Tile que generalmente son extraídos desde un servidor web, este tipo de capa es el utilizado para mostrar los datos de OSM. Por el contrario, en Leaflet solo debemos añadir una capa nueva y la librería se encarga de configurar cada elemento dentro de ella para que se visualice adecuadamente.

React

Para crear la interfaz hemos decidido utilizar React pues esta librería es capaz de trabajar junto a cualquier otra librería escrita en JavaScript a diferencia de otras que solo permiten utilizar librerías desarrolladas especialmente para trabajar con ellas. No obstante, hacer que React y Open Layers pudieran trabajar juntos no ha sido tarea fácil pues se deben sincronizar las actualizaciones que ambas librerías realizan sobre el DOM. A pesar de ello, se ha logrado que componentes de ambas librerías puedan cooperar entre ellos sin limitación alguna.

La ventaja de utilizar React frente a no haberlo hecho es la capacidad de poder reutilizar componentes de forma sencilla, así como de poder modificar la interfaz de forma dinámica según se utiliza y sin necesidad de actualizar la página entera.

Esto puede apreciarse especialmente en que los componentes utilizados en la barra lateral son los mismos que los utilizados en el pop up que aparece cuando el usuario selecciona los elementos de mapa. Esto nos ha permitido en definitiva crear la aplicación más rápidamente.

Conexión con la base de datos

La conexión con la base de datos desde la interfaz no se hace de forma directa si no que se realiza a través del servidor REST que hemos creado encima de ella. Esto es de gran utilidad por múltiples razones:

- Las peticiones que se pueden realizar sobre la base de datos están más controladas.
- No es necesario conocer las consultas SQL concretas que se realizan sobre la base de datos, su estructura es opaca desde la aplicación web.
- Tener un servidor escrito en un lenguaje de programación completo permite realizar acciones más complejas que las peticiones SQL por si solas como formatear los datos o pre procesarlos antes de enviarlos a quien los solicite.

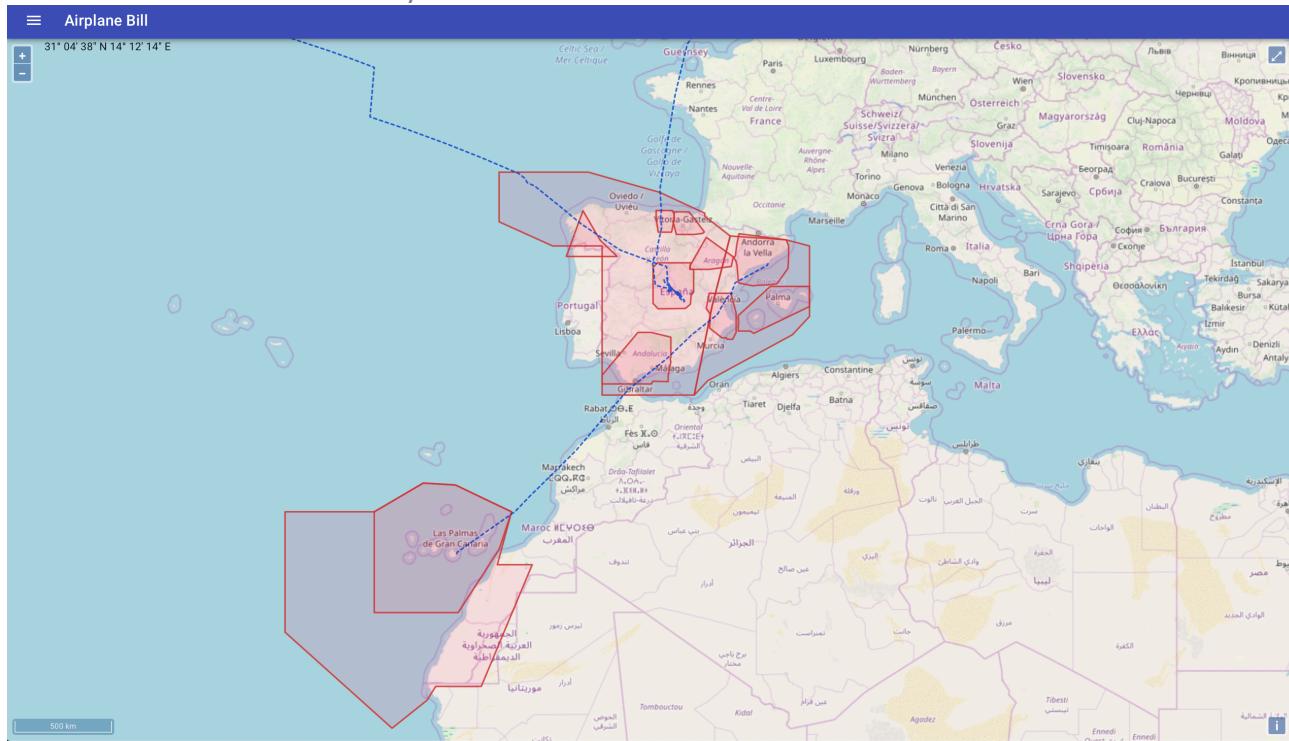
Para realizar las peticiones se ha utilizado la librería axios.

Acceso a la interfaz

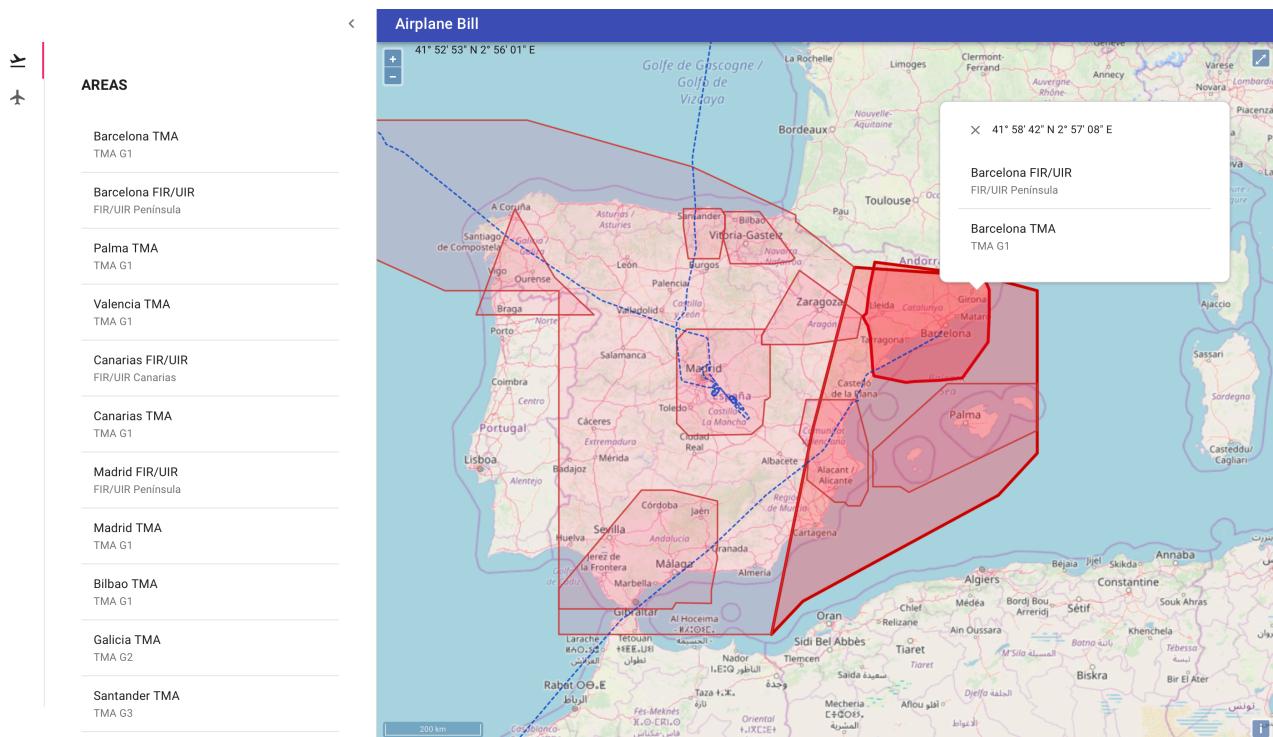
React proporciona un servidor por defecto muy útil para desarrollar la página web pero recomiendan que no debe ser utilizado en producción. Entre otras cosas esto es así pues en caso de que se produzca algún error del que en condiciones normales JavaScript podría recuperarse, la aplicación fallará por completo alertando al desarrollador de la existencia del problema.

Para evitar esto la aplicación ha sido transformada en un único archivo html como si de una página no desarrollada con React se tratara. Posteriormente dicho archivo es proporcionado a un servidor NGINX que se encargará de servirlo como un documento estático a todos los usuarios que tengan acceso a él y escriban su dirección en el navegador.

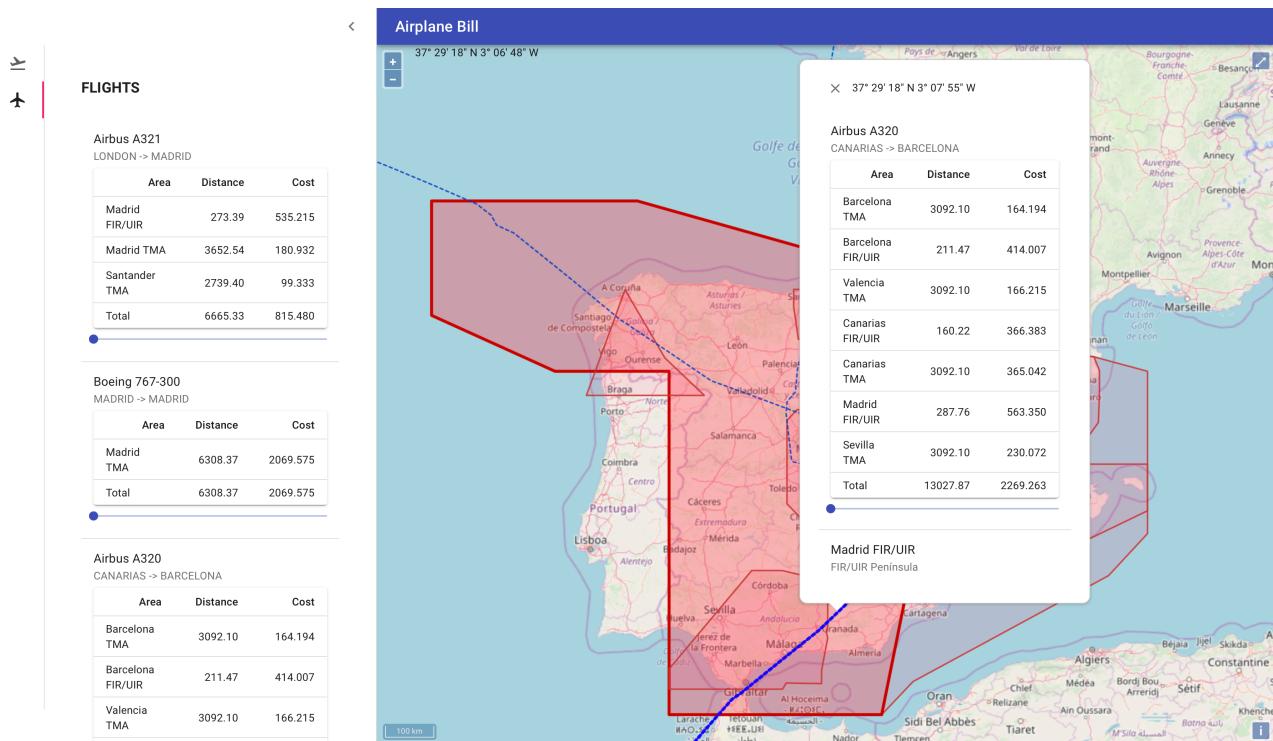
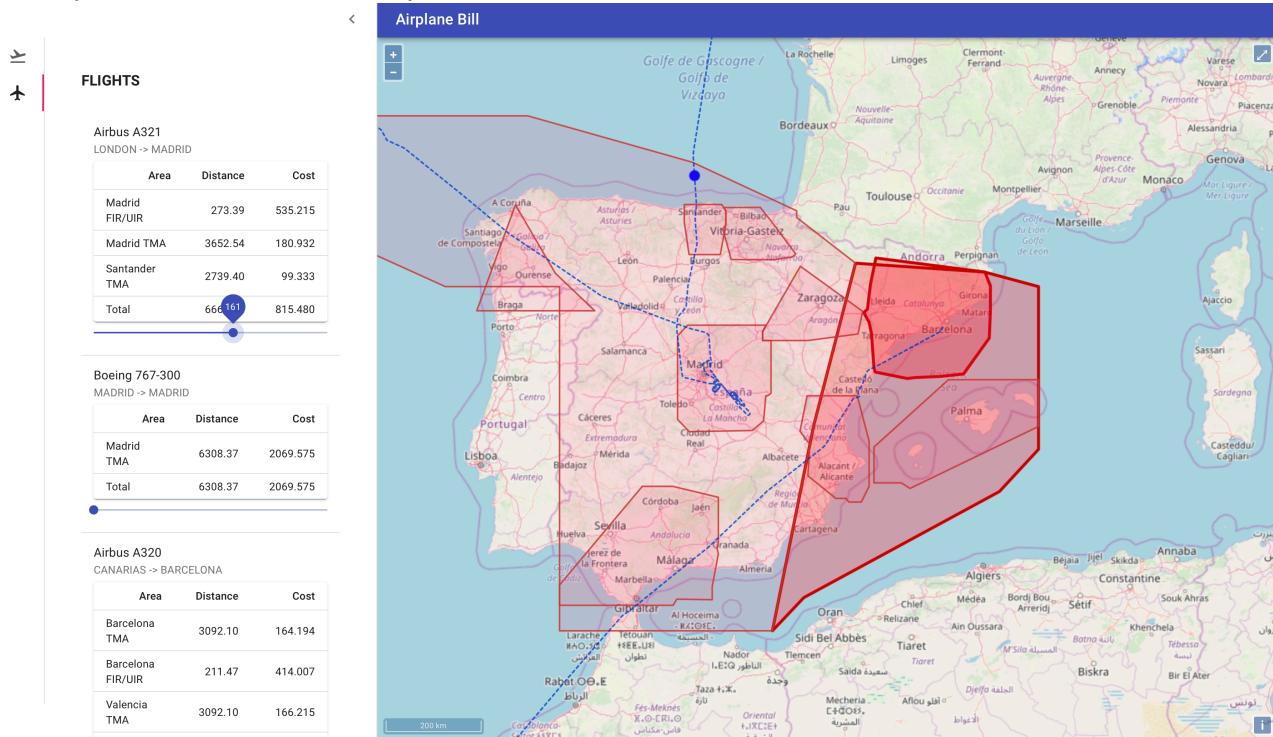
Interfaz web mostrando las rutas y las áreas.



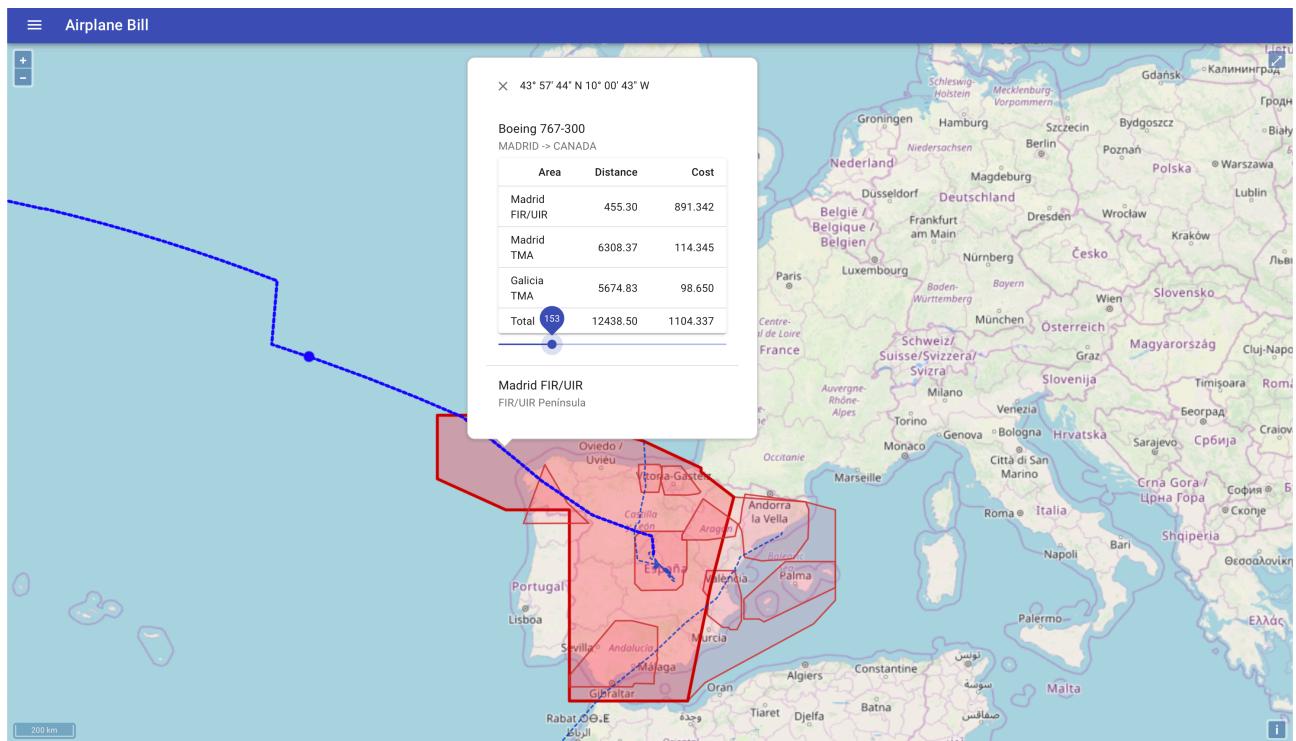
Cuando las áreas o las rutas se seleccionan cambian de color, sus datos aparecen en el pop up.
En el panel lateral podemos ver el listado de todas las áreas del mapa.



En el panel lateral podemos ver el listado de todos los vuelos.
 De ellos vemos el coste del vuelo desglosado por áreas y en total.
 También podemos utilizar el slider para simular el avión recorriendo la ruta.



Podemos ver también toda esta información de los vuelos el pop up al seleccionar una ruta.



OTRAS FORMAS DE VISUALIZAR LOS DATOS

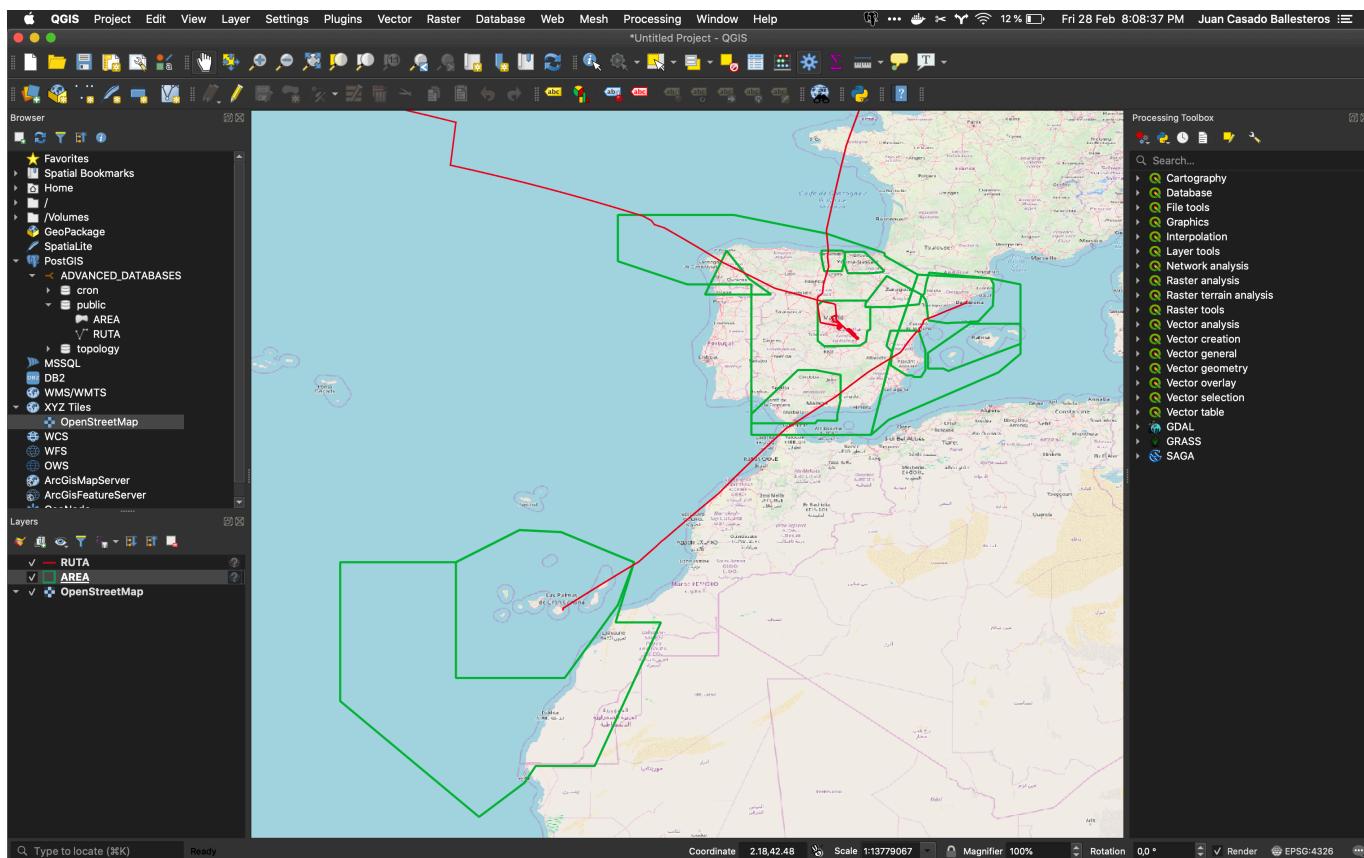
En nuestro caso hemos decidido crear una interfaz web para visualizar los datos y mostrar el coste de las rutas de los aviones, no obstante, esto podría no haberse realizado. Otra opción para acabar obteniendo estos mismos resultados o muy similares hubiera sido no haber creado ningún tipo de interfaz gráfica y haber hecho una sencilla aplicación de consola. Sin embargo, existen herramientas que ya proporcionan una interfaz gráfica preparada para visualizar datos espaciales cuya configuración es casi inmediata.

QGIS

En nuestro caso hemos decidido utilizar QGIS como ejemplo de una de estas herramientas. QGIS es capaz de conectarse directamente a la base de datos y extraer de ella toda la información visualizándola posteriormente sobre un mapa interactivo.

En QGIS podemos incluso visualizar los mismos mapas de OSM que han sido utilizados en la interfaz web.

En la captura mostramos QGIS conectado a nuestra base de datos mostrando las rutas almacenadas en rojo, las áreas en verde y de fondo un mapa de XYZ extraído de OSM.



DESPLIEGUE DE LA APLICACIÓN

Uno de los problemas recurrente a la hora de trabajar con bases de datos, servidores y en especial con Java son la instalación y la existencia de múltiples versiones del mismo programa. Es habitual tener una JDK configurada para un conjunto de aplicaciones pero que al descargar una nueva requiera una versión distinta que acaba sobrescribiendo o reemplazando las que ha existía. También es habitual que distintas versiones de la misma base de datos tengan problemas de compatibilidad con ciertas herramientas y de que no todas las versiones estén disponibles para todos los sistemas operativos.

Para solucionar todos estos problemas hemos decidido utilizar contenedores en los que poder encapsular versiones concretas y compatibles de todos los elementos que utilizamos.

Docker

Hemos utilizado Docker para construir todas las partes que conforman nuestra aplicación. Cada una de ellas estará encapsulada en su propio contenedor, aisladas entre ellas y aisladas también de la máquina en la que se ejecuten con solo los puertos necesarios como forma de acceso a los contenedores.

Esto nos permite automatizar el proceso de construcción de la aplicación con garantías de que esta podrá funcionar en cualquier máquina en la que sea ejecutada.

Como apunte adicional comentaremos la forma en el que en contenedor que contiene la base de datos PostgreSQL+postgis se construye. Para hacerlo utilizamos la versión 12 de PostgreSQL con su extensión postgis compatible ambas descargadas del repositorio **kartoza** especializado en aplicaciones GIS. Posteriormente solo debemos configurar la base de datos, el usuario y la contraseña con la que deseamos acceder a ella. Finalmente indicaremos el puerto desde el que deseamos acceder a la base de datos.

Hemos creado también un volumen, lo cual es algo opcional, para que la base de datos guarde la información que contiene, aunque el contenedor finalice su ejecución.

Mostramos los tres contenedores que se utilizan (postgis, tomcat y web) en funcionamiento.

~/Documents » docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
077d2f19ce6f9	advanced_databases_postgis	"bin/sh -c /docker-..."	About an hour ago	Up About an hour	0.0.0.0:5432->5432/tcp	advanced_databases_postgis_1
52d3c686309	advanced_databases_web	"nginx -g 'daemon off;"	About an hour ago	Up About an hour	0.0.0.0:80->80/tcp	advanced_databases_web_1
cf663660d40d	advanced_databases_tomcat	"catalina.sh run"	About an hour ago	Up About an hour	0.0.0.0:8080->8080/tcp	advanced_databases_tomcat_1