




12 DE ENERO DE 2023

DEVELOPING A DATALAKE

DESARROLLO DE APLICACIONES PARA CIENCIA DE DATOS

GRADO EN CIENCIA E INGENIERÍA DE DATOS, 2º

Escuela de Ingeniería Informática, Universidad de Las Palmas de Gran Canaria





## Resumen

Este trabajo consiste en aplicar e implementar los conocimientos adquiridos en clase referidos a la arquitectura lambda y tratamiento de eventos. Por esta misma razón, este proyecto consiste en obtener datos periódicos de la “AEMET” e introducirlos en un datalake. Cada día, se debe recorrer el datalake e introducir las medidas del aire máximas y mínimas en una base de datos Sqlite. De forma paralela, se debía desarrollar una API que proporcionase dicha información. La aplicación está dividida en tres módulos independientes y que, por lo tanto, pueden ser ejecutados por separado: un feeder, que se encarga de obtener los datos de la Aemet e introducirlos en el datalake en función de la fecha de la medición (hay que tener en cuenta que cada petición debe ser tratada como un evento); un datamart-builder, que diariamente se encargará de leer el datalake y de guardar las temperaturas máximas y mínimas diarias; y un temperature-api, que será el encargado de dar soporte a la API REST. A lo largo de esta memoria, se irá explicando más en profundidad la forma en la que trabaja cada módulo.

La aplicación no se puede decir que sea rápida, ni eficiente, ni que esté al 100% optimizada, aunque he intentado que, con mis conocimientos actuales, ésta no incurra en acciones innecesarias o tenga código zombie; por ende, se puede decir que, aunque no será la más rápida, si se puede decir que realiza bien su función.

## Índice

Resumen.....	2
Recursos Utilizados .....	2
Diseño.....	3
Conclusiones .....	6
Líneas Futuras .....	6
Bibliografía .....	6

## Recursos Utilizados

Para la realización de este proyecto se ha utilizado el entorno de programación “IntelliJ Idea Community Edition 2022.2.3”, así como las dependencias de gson para serializar la información y devolverla por la API y para deserializar la información proporcionada por la AEMET; jsoup para hacer el “scrapping” de los datos; spark-core para la implementación de la API; y sqlite-jdbc para el manejo del datamart. Para el control de versiones se ha usado Git, y para la realización de esta memoria se ha utilizado “Microsoft Word”.

## Diseño

Como se dijo en el resumen, la aplicación está dividida en los siguientes módulos independientes:

### Feeder

Es el módulo que se encarga de tomar los datos de la API de AEMET e introducirlos en el datalake. Aunque actualmente hay que darle a correr de forma manual cada vez que se quieran recoger los datos, existe un código comentado que hace que, con darle una vez, descargue los datos cada 60 minutos. Las diferentes clases de este modelo son:

- Interfaz ApiConsumer: define los métodos que las implementaciones deben (valga la redundancia) implementar, tales como hacer una petición, conseguir datos, o filtrar los datos por ubicación.
- AemetApiConsumer: implementación de la interfaz definida en el anterior punto. Tiene un atributo de instancia que es la apiKey. Y a la hora de tomar los datos, convierte la respuesta en un array de Strings (donde cada elemento será una medida en json), que luego convertirá a un array de Measure, mediante el método fromJson(). Para filtrar las medidas, se le deberá pasar como parámetro una lista de medidas y un mapa con las coordenadas que definen el espacio a filtrar.
- Interfaz Datalake: define los métodos write and read.
- FileDatalake: implementación de Datalake. Tiene un atributo de instancia, que es la carpeta donde se van a guardar las medidas diarias. Para el método write, dado un array de Measure, se realiza un stream donde para cada medida, se guarda en el datalake. El destino del archivo de la medida se obtiene a partir de la fecha de la misma; por esto mismo, la función getArchiveName() se encarga de obtener la ruta del archivo. Una vez tenemos el archivo abierto, se pasa la medida a json (que es el formato en el que se guarda en el datalake) y hace las siguientes comprobaciones: si la medida a introducir se encuentra ya en el datalake, no se escribirá; por el contrario, si no se encuentra, solo entonces se escribirá en el datalake. De esta forma, no dependemos de cada cuanto se realice las peticiones a AEMET, ni tampoco tendremos que preocuparnos por si la aplicación se para, pues podremos recuperar hasta un máximo de 23 horas. Algunas estaciones no envían mediciones todas las horas (por razones que desconozco), pero luego recuperan, por lo tanto, con este formato podemos adaptarnos a esta “anomalía” que tienen algunas estaciones. Por último, me gustaría añadir que estoy desechando aquellas mediciones que tienen una temperatura de 0.0 ya que es imposible que en San Cristóbal la temperatura fuese de 0.0°C a las 11 de la mañana.
- Controller: controlador del módulo que, como su nombre indica, controla que todos los procesos se realizan de forma correcta y en el orden correspondiente.

### Datamart-builder

Este módulo se encarga de leer el datalake y de generar dos tablas donde se almacenará la información relativa a las estaciones donde se produjeron las temperaturas máximas y mínimas de cada día. Se supone que el datamart se debería poner a trabajar cada noche, pero no es automático, ya que dejar 23 horas el programa en espera no tiene mucho sentido. De forma que cada vez que quieras “actualizar” el datamart, debes darle a correr. Digo actualizar entre comillas, porque esta implementado de forma que cada vez que corre la aplicación, el datamart se elimina y se vuelve crear, por lo que podemos abstraernos de cada cuanto se debe actualizar el datamart,

así como de qué mediciones ya hemos tenido en cuenta y cuáles no. Las clases que conforman este modelo son:

- Interfaz DataBase: define los métodos createTable() y addRegister().
- SQLiteDatabase: implementa la interfaz definida anteriormente y tiene tres atributos de instancia (ruta base de datos, con y statement), pero solo se le pasa un parámetro por constructor (la ruta donde se va a crear la base de datos). En el constructor, se crean tanto las conexiones como el statement, mediante la función connect(). El método createTable() crea una tabla a partir del nombre y los campos pasados por parámetro. Por último, el método addRegister(), como bien su nombre indica: introduce un registro en la tabla dados el nombre y la medida a introducir en la tabla.
- Interfaz Datalake: define los métodos write() y read().
- FileDatalake: implementación de Datalake. El método write() ya está explicado, aunque este módulo no lo usa. El método read() lee todas las medidas de un archivo dada el nombre del archivo que se quiere leer, y devuelve una lista de medidas.
- Controller: dado el directorio, comprueba que se trata de un directorio y por cada archivo que hay en el directorio, lee todos los nombres de los archivos que hay en el directorio, y para cada uno, manda a leer dicho archivo al FileDatalake, mediante dos streams se obtienen las medidas que contienen las temperaturas máximas y mínimas, y luego ordena al SQLiteDatabase que lo introduzca en la tabla correspondiente.

### Temperature-api

Este módulo se encarga de gestionar la API mediante la cual se proporcionarán los datos de temperaturas máximas y mínimas recogidas en el datamart. Las dos peticiones a gestionar son las siguientes: get(v1/places/with-max-temperature) y get(v1/places/with-min-temperature). Ambas admiten por parámetros de query la fecha de inicio y la fecha hasta la cual se quieren obtener los datos, las cuales se deben introducir según el formato AAAA-MM-DD. En este apartado, he hecho las siguientes comprobaciones: si no se introduce ese formato, se hace halt de un 400 con el mensaje “no date provided”; de igual manera, si solo se introduce una fecha saltará la misma excepción. Por último, si he manejado el hecho de que puedan venir en orden inverso (fecha de inicio como fecha final y fecha final como fecha de inicio) por equivocación del usuario. En tal caso, se intercambian los parámetros y listo. En este trabajo, al igual que en el del Scrapper, he decidido inyectarle a la instancia del WebService un Controlador, de forma que pueda pedirle los datos al mismo mediante métodos de instancia. A continuación, se procederá a explicar las siguientes clases e interfaces que conforman el modelo.

- Interfaz DataBase: define los métodos readRegister() ya que la única operación que este módulo necesita hacer sobre el datamart es leer.
- SQLiteDatabase: implementación de DataBase, donde dada una fecha y el nombre de la tabla, devuelve un ResultSet de las coincidencias encontradas (en este caso solo devuelve una coincidencia ya que la fecha es primary key).
- Controller: posee un atributo de instancia que es el path a la base de datos, y contiene además dos métodos de instancia mediante los cuales la API le pide los datos: getMinTemperature y getMaxTemperature. Donde dadas dos strings (fechas), mediante un for va ordenando a la instancia de SQLiteDatabase que vaya sacando datos del datamart y metiéndolos en una lista de Measures. Me hubiera gustado haber usado el parámetro BETWEEN para la obtención de los registros, pero al ser fechas el parámetro de búsqueda, no tenía muy claro si fuese a funcionar.
- SparWebService: gestiona la API y se encarga de comprobar que los parámetros son correctos, aunque no tengo del todo claro si la comprobación de los parámetros debe ir aquí o en el Controlador, porque es cierto que la API deberá ser independiente y abstraerse de ese tipo de cosas, pero por otro lado, solo comprueba que lo que se le

introduce es una fecha y si se le introducen las dos. Me parecería una pérdida de tiempo pasarle datos incorrectos al controlador pudiendo comprobarlo de manera muy sencilla en la misma API.

## Conclusiones

Igual que en otros proyectos, también me gustaría aprovechar esta ocasión para contar mi experiencia personal. Este proyecto me ha parecido el más interesante sin duda de los tres que hemos realizado en el curso (y a la vez el más complejo) pues incorpora tres módulos independientes y ejecutables, por lo que la carga de este trabajo es considerablemente mayor a los demás, y por lo tanto, tiene sentido que valga más que los demás. Sin embargo, no me ha parecido extremadamente complicado o imposible de realizar (gracias en mayor parte a las explicaciones del profesorado), sino, como dije antes: largo. Este trabajo también ha servido como recopilación de los anteriores, al incorporar tanto manejo de base de datos, como desarrollo de un servicio web RESTful, por lo que lo “único” nuevo que ha incorporado este trabajo, ha sido la gestión de eventos mediante un datalake y un datamart. El datamart fue sencillo de implementar, pero he de admitir que tuve mis más y mis menos con el datalake, pues tenía ciertas dudas sobre como gestionar lo de “recoger información cada hora” y al final me decanté por comprobar si cierta medida estaba ya o no en el datalake, lo cual se dijo en clase y es mucho más fácil que depender de cada cuánto tiempo se recogen datos. Aparte de eso, ha sido un trabajo llevadero y entretenido de hacer, y gracias al cual he podido entender de primera mano como funcionan las arquitecturas lambda.

Aprovecho también, ya que este es el último trabajo, para dar mi opinión general sobre la asignatura. Pienso que los contenidos de los que se han hablado en clase han sido muy interesantes y no me ha importado en absoluto que en ocasiones nos saliésemos del proyecto docente. El transcurso de las sesiones (prácticas y teóricas) también han sido bastante correctas, así como la labor del personal docente. Asimismo, los trabajos me han parecido muy interesantes los tres, y opino que la elección ha sido bastante acertada. Sin embargo, me hubiera gustado que en clase se hubiese dado una pequeña introducción al Scrapping, ya que es algo que eché un poco de menos. Yo supongo que se dio por hecho que, como habíamos hecho anteriormente un Scraper en Fundamentos del Marketing y Comportamiento del Consumidor, ya teníamos cierta agilidad en este tema; pero al menos por mi parte no lo siento así, pues en dicha asignatura, el Scrapping podría ser realizado en cualquier lenguaje de programación y con cualquier herramienta que encontrásemos en Internet, sumado a que el docente tampoco nos dio ninguna introducción al Scraper. Yo hice el Scrapping con Python y Selenium, y pasar a Java y a Jsoup fue un poco duro ya que, aunque mantenían ciertas estructuras, muchas otras cosas eran diferentes y prácticamente, lo anteriormente aprendido me sirvió de poco. Por último, me gustaría dar las gracias por los seminarios que se ofrecieron al principio de curso (pues eran bastante necesarios, ya que al principio de curso andábamos-o yo al menos-bastante perdidos) y me hubiese gustado dar alguno que otro más en la recta final del curso (aunque entiendo que después del incidente, sumado a que la coincidencia de horarios era muy complicada, fuese muy complicado).

## Líneas Futuras

Mirando a líneas futuras, tal vez sería buena idea obtener otro tipo de información meteorológica como previsiones; también ampliar nuestras fronteras a otros países, intentar usar otras fuentes meteorológicas...

Y ya con esto llegamos al fin de la memoria, esperando que no se le haya hecho extremadamente larga y agrediéndole su tiempo.

## Bibliografía

<https://chat.openai.com/chat> (para dudas, no ha generado código)

<https://opendata.aemet.es/dist/index.html>

