JSON y servicios web con Java

Práctica 2 de laboratorio

Aplicaciones Telemáticas
ETSIT-UPV

J. E. López, F. J. Martínez Zaldívar

Índice

1. Introducción y objetivos

Los servicios web son utilizados hoy en día en gran cantidad de aplicaciones. En la presente práctica nos planteamos como objetivos ser capaces de, empleando el lenguaje de programación Java, utilizar estos servicios elaborando peticiones de tipo API REST, recibir los resultados y extraer de los mismos la información en la que estemos interesados.

Para ello, utilizaremos un ejemplo de entre los múltiples que podemos encontrar en la red: el servicio OpenData de AEMET, la Agencia Estatal de Meteorología. Pediremos la predicción meteorológica para ciertos intervalos o instantes horarios de un municipio en concreto y extraeremos ciertas propiedades de dicha predicción, escribiéndolas en pantalla.

Adicionalmente será necesario trabajar con certificados digitales para poder acceder a estos servicios web, por lo que aprovecharemos la ocasión para aprender a obtener dichos certificados, almacenarlos y utilizarlos convenientemente.

1.1. Formación de grupos

Los grupos para realizar esta práctica serán de uno o dos alumnos, intentando mantener los de la práctica anterior. Si por algún motivo hubiera algún cambio, este debe comunicarse al profesorado. Se sugiere que se lean detenidamente todas las indicaciones dadas en la presente memoria para evitar que un incumplimiento en alguna parte de la misma pueda mermar el resultado de la calificación.

1.2. Software necesario y comunicaciones

El software necesario consistirá en:

- Editor de textos plano
- JDK
- Paquetes de Java `javax.json`, `okhttp` y `okio`, proporcionados junto con la memoria de la presente práctica

Respecto a comunicaciones será necesario tener evidentemente conexión a internet.

2. Servicio AEMET OpenData

AEMET es la Agencia Estatal de Meteorología. *"El objeto de AEMET, según el artículo 1.3 del Real Decreto 186/2008, de 8 de febrero por el que se aprueba su Estatuto, es el desarrollo, implantación, y prestación de los servicios meteorológicos de competencia del Estado y el apoyo al ejercicio de otras políticas públicas y actividades privadas, contribuyendo a la seguridad de personas y bienes, y al bienestar y desarrollo sostenible de la sociedad española".* (Véase `http://www.aemet.es`.)

Asimismo, *"AEMET OpenData es un API REST (Application Programming Interface, REpresentational State Transfer) a través del cual se pueden descargar gratuitamente los datos explicitados en el Anexo II de la resolución de 30 de diciembre de 2015 de AEMET".* (Véase

`http://www.aemet.es/es/datos_abiertos/AEMET_OpenData`

para más detalles.)

Una API REST define una serie de funciones y sus detalles de uso, que pueden utilizarse para realizar peticiones por parte de un cliente, y recibir respuestas de un servidor, vía el protocolo HTTP. La API debe explicitar los parámetros de la petición, el formato de la respuesta, limitaciones, uso público o con claves de API, métodos HTTP (GET/POST/PUT/DELETE), etc.

En el caso que nos ocupa, emplearemos la API REST de AEMET que permite acceder a lo que la agencia califica como datos abiertos. La utilización de este servicio requiere poseer una clave y utilizarla adecuadamente en el momento de realizar la petición.

Ábrase la página `http://www.aemet.es`, seleccionando en el menú la opción `Datos abiertos` (parte superior derecha de la página web) y después `AEMET OpenData`, tal y como muestra la figura 1.

Váyase entonces al final de la página y selecciónese `AEMET OpenData`, tal y como se muestra en la figura 2.

Tras lo cual, se nos mostrará la página tal y como aparece en la figura 3, donde podemos observar en la parte inferior cuatro bloques:

- AEMET OpenData: ¿Qué es?

  Donde se explica el servicio, sus características y cómo operar con él. Se sugiere que se lea detenidamente toda esta información.

- Obtención de API Key: Solicitar

  Introduciendo una dirección de correo electrónico, se recibirá un correo en el que se deberá confirmar la generación de la clave (pulsando en el enlace `Confirmar generación API Key` dentro del correo recibido),

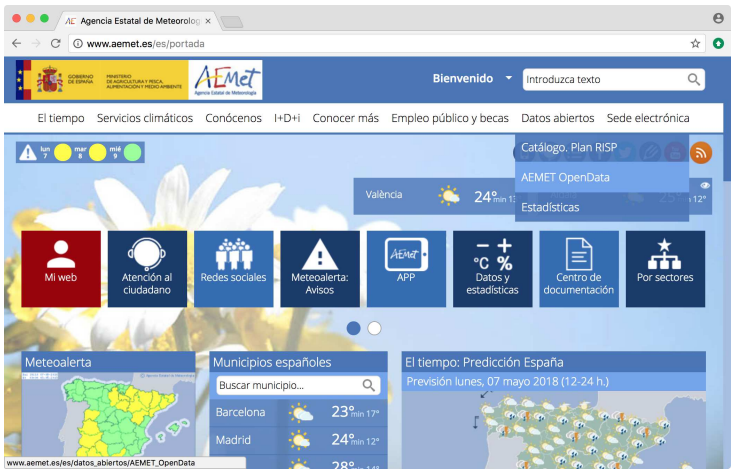

Figura 1: Página principal de AEMET

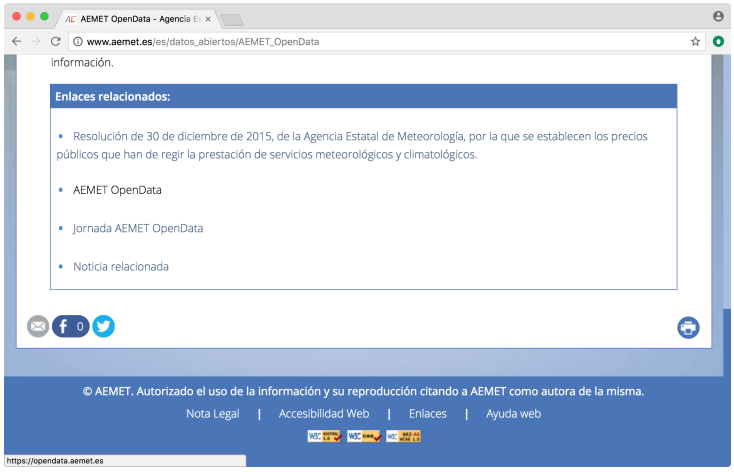

Figura 2: Selección de AEMET OpenData

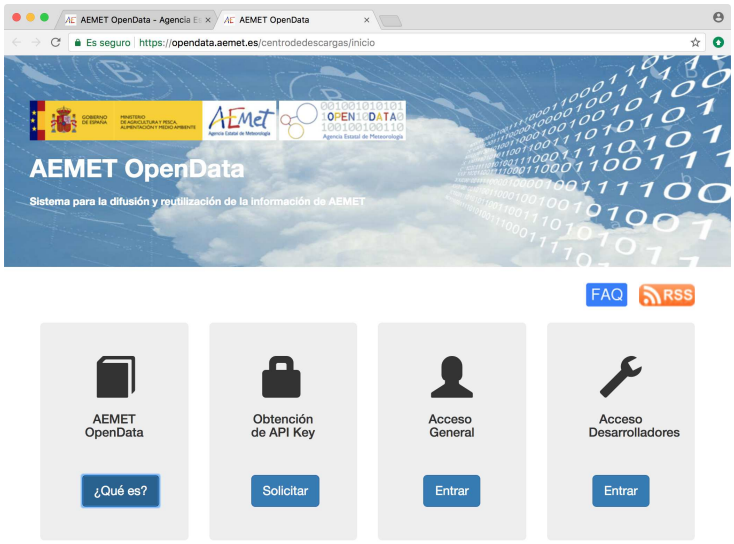

Figura 3: Selección de opciones en AEMET OpenData

tras lo cual se recibirá de nuevo otro correo electrónico en el que se apreciará claramente la clave necesaria. Es conveniente tenerla a mano, por ejemplo, copiándola en el portapapeles para utilizarla posteriormente. Esta clave es efímera, por lo que requeriría su renovación transcurrida cierta cantidad de tiempo.

- Acceso General: Entrar

  Es una interfaz web cómoda con la que realizar cierto tipo de pruebas y consultas. Puede ser ilustrativo hacer algunas pruebas para entender la forma en la que se elabora la petición.

  Para poder usar la interfaz será necesario pegar la clave recibida por correo electrónico en el cajetín titulado **Introduzca su API Key:** ubicado en la parte superior de la página (véase la figura 4).

  Posteriormente puede elegirse qué tipo de datos queremos observar. Se sugiere, como ejemplo, que se seleccione el que aparece en la misma figura 4, dentro del primer bloque titulado *Observación convencional*, denominado *Datos de observación. Último elaborado*; elíjase, por ejemplo, `València/Valencia` y la estación meteorológica ubicada en el aeropuerto (`8414A - Valencia Aeropuerto`). Tras pulsar `Obtener`, obtendremos en una ventana de navegación independiente similar a lo que se muestra en la figura 5.

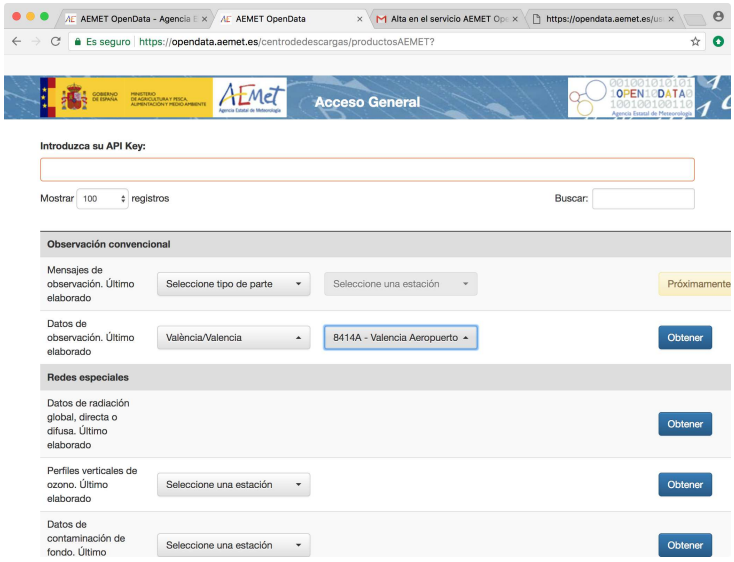

Figura 4: Acceso general en AEMET OpenData

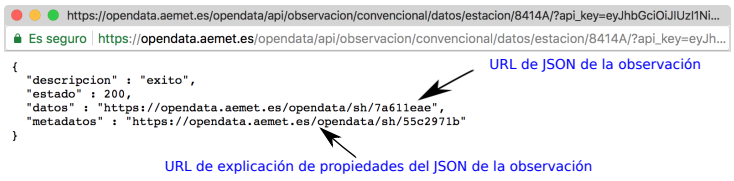

Figura 5: Respuesta JSON del servicio de *Datos de observación. Último elaborado*

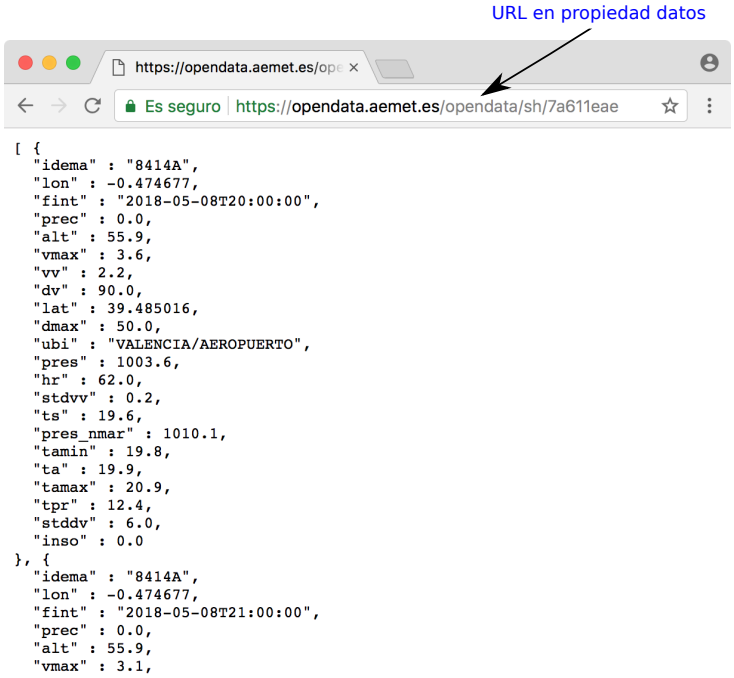

Figura 6: Respuesta JSON del la URL contenida en la propiedad `datos`

En esta respuesta, con formato JSON, encontramos una propiedad efímera (al cabo de unos 5 minutos ya no será válida) denominada `datos` que contiene una URL tal que utilizada, por ejemplo en un navegador convencional, nos entregará el resultado requerido, tal y como se muestra en la figura 6, pudiendo apreciar claramente el nombre y el valor de ciertas propiedades.

Podemos encontrar una breve descripción de cada una de las propiedades que aparecen en la respuesta JSON de la figura 6, en la URL indicada en la propiedad `metadatos` de la figura 5, tal y como se muestra en la figura 7.

Recuérdese que estas URL expiran al cabo de cierto tiempo por lo que no son reutilizables de manera indefinida.

Se sugiere que se ejecuten algunos de estos servicios para observar la información entregada por cada uno de ellos.

Puede ser un poco incómodo apreciar la estructura JSON de la información descargada por lo que se sugiere que se copie en el portapapeles y se pegue en algún visor/validador JSON como por ejemplo `http://jsonviewer.stack.hu/`, tal y como puede apreciarse en la fi-

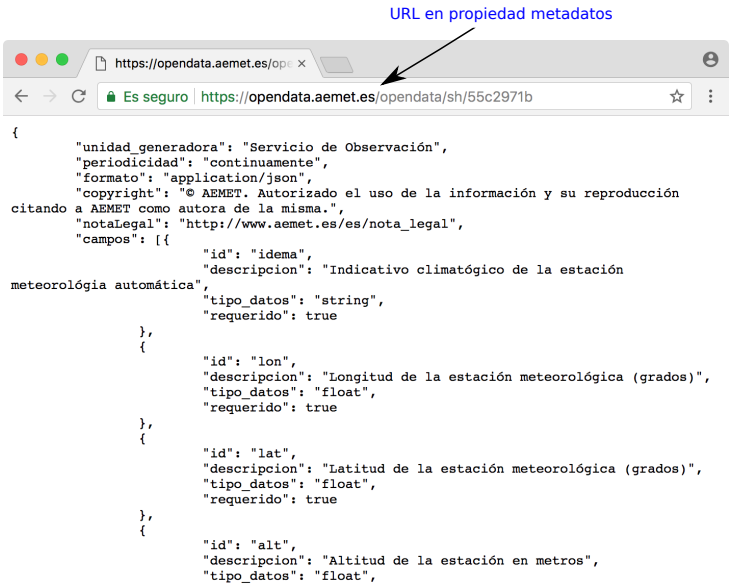

Figura 7: Respuesta JSON del la URL contenida en la propiedad `metadatos`

gura 8, donde puede observarse claramente que la respuesta consiste en un array JSON, donde cada una de sus componentes es un objeto en el que se muestran claramente sus propiedades.

- Acceso Desarrolladores: Entrar

  En este bloque encontramos tres subbloques, tal y como se muestra en la figura 9.

  - "Documentación AEMET OpenData. HATEOAS. Acceso a documentación dinámica AEMET OpenData y autodescubrimiento HATEOAS, que permite conocer los recursos del API y sus detalles."

    Aquí se desarrollan todos los servicios junto con detalles de la API asociada. Por ejemplo, se sugiere que se intente observar cuál es la predicción diaria para un municipio elegido por el alumno. Para ello, selecciónese el bloque `predicciones-específicas` y concretamente la petición indicada con el método GET y la URL `/api/prediccion/especifica/municipio/diaria/{municipio}` Debe introducirse el código de municipio en el cajetín **municipio**. El código de municipio **NO** es el código postal. Para obtenerlo, puede descargarse un fichero correspondiente a una hoja de

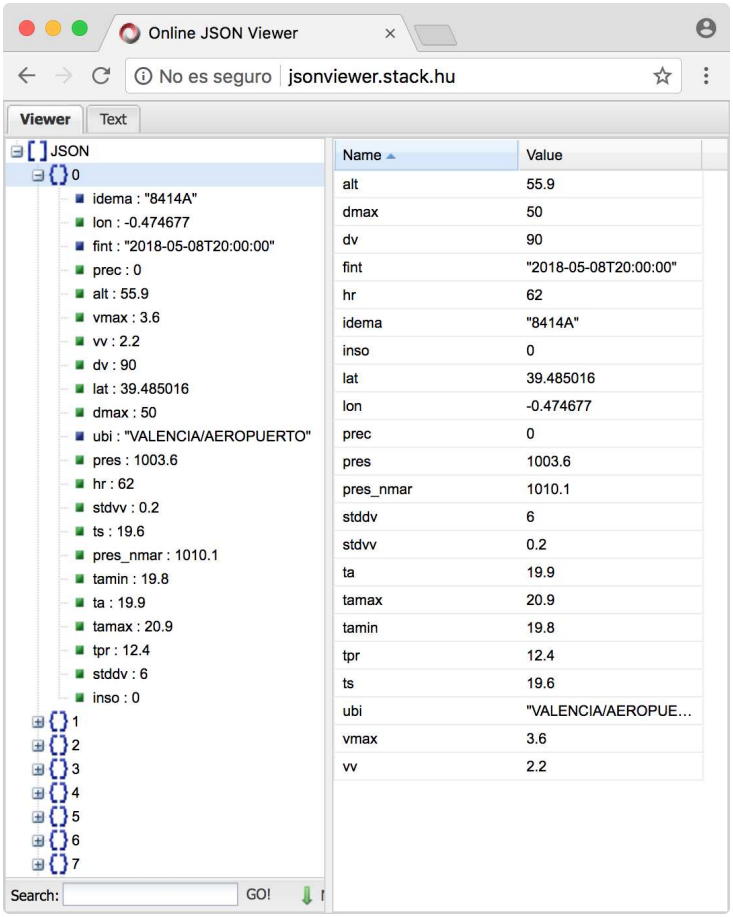

Figura 8: Visor JSON

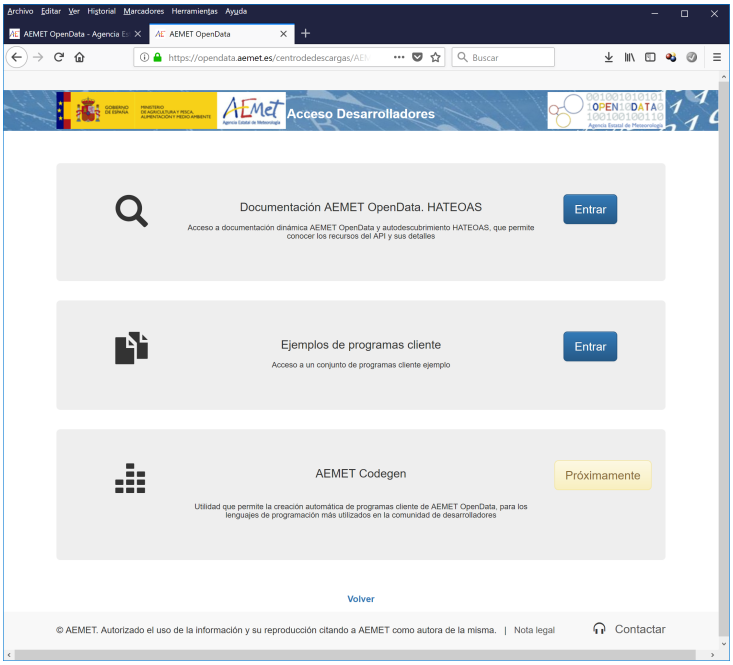

Figura 9: Acceso a desarrolladores

cálculo, tal y como se muestra en el enlace ubicado a la derecha del cajetín de **municipio:**, denominado **código de municipio** (véase la figura 10). El código consta de la concatenación de los números (*strings* realmente) de las columnas **CPRO** y **CMUN** de dicha hoja de cálculo. Así por ejemplo, el código de municipio de Ademuz de Valencia sería `"46001"` (véase la figura 11) y el de Alburquerque de Badajoz sería el `"06006"`.

Tras darle al botón `Try it out!`, ubicado en la parte inferior de la sección desplegada de la predicción diaria por municipio, se nos mostrará la URL de la petición en el cajetín denominado `Request URL`:

```
https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/46001
```

A esta URL habrá que añadirle el parámetro de la API key de la siugiente forma:

```
https://opendata.aemet.es/opendata/api/prediccion/especifica/municipio/diaria/46001?api_key=xxxxxx....xxxx
```

obteniendo de nuevo un objeto JSON con las propiedades efímeras `datos` y `metadatos` de las que extraer la información necesaria para realizar la consulta definitva.

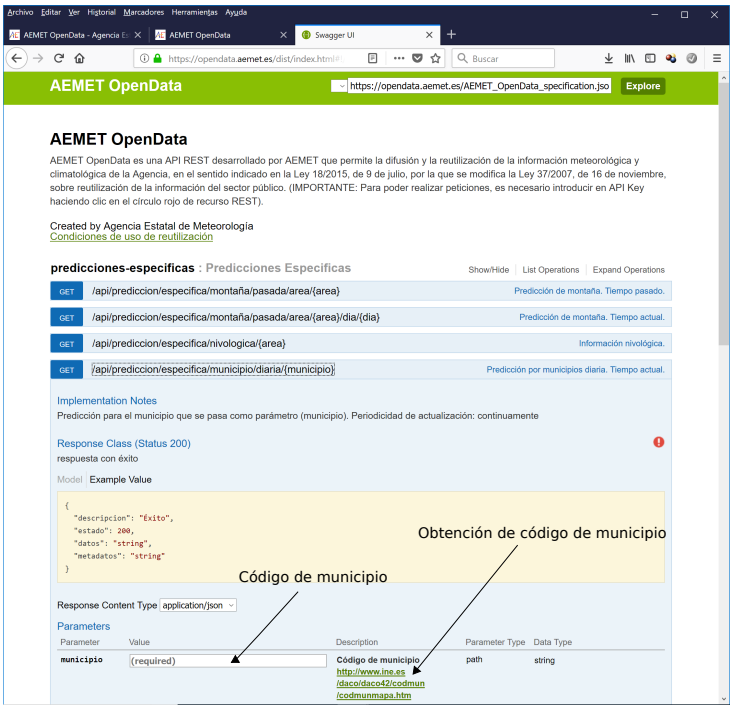

Figura 10: Documentación AEMET OpenDATA. HEATEOAS.

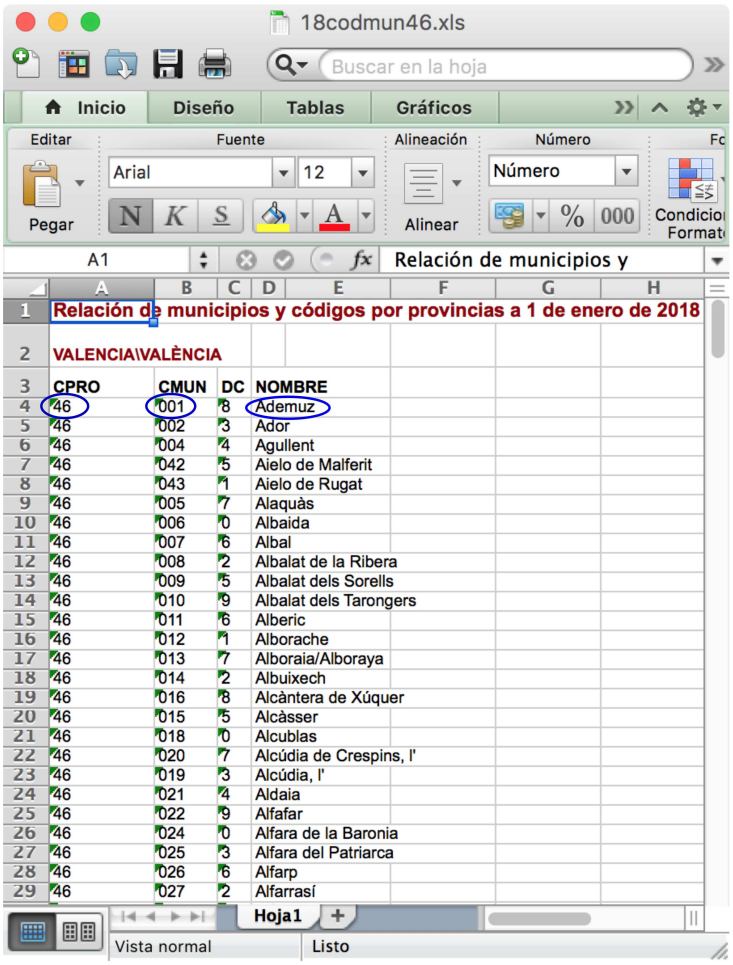

Figura 11: Códigos de municipios de la provincia de Valencia.

- Ejemplos de programas cliente. Acceso a un conjunto de programas cliente ejemplo.

  Si pulsamos el bloque asociado a Java, nos encontramos con algunos *snippets* o pequeños códigos reutilizables, que en nuestro caso es:

```
OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
  .url("https://opendata.aemet.es/opendata/api/valores/"
       +"climatologicos/inventarioestaciones/"
       +"todasestaciones/?api_key=...")
  .get()
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

  Aquí observamos dos clases necesarias para operar: `OkHttpClient` y `Request` (pertenecientes ambas al paquete `okhttp3`, el cual, a su vez requiere el paquete `okio`), las cuales no aparecen en las distribuciones clásicas de Java de Oracle. Pueden ser descargadas de sitios de confianza, o bien utilizarse las que se proporcionan junto con la documentación de la presente práctica.

  Puede intuirse que las instrucciones del anterior *snippet* configuran la URL de la petición, la efectúan y reciben la respuesta. A partir de la respuesta HTTP debe extraerse el contenido del cuerpo o *body* en formato *string*, lo cual puede conseguirse como:

```
String respuesta = response.body().string();
```

  A partir de aquí, habría que realizar el oportuno análisis JSON y extraer las propiedades en las que estamos interesados.

3. JSON y el paquete `javax.json` de Java

La intención final de la presente práctica es automatizar el proceso de peticiónes REST de información y extracción de propiedades a partir del objeto JSON recibido. Es decir, ya no emplearemos los navegadores para descargar y visualizar información en formato JSON, sino que la elaboración de la petición REST, su envío, la recepción de la respuesta y la extracción de la información de la misma, y su reiteración, se hará todo en Java.

Ya hemos visto anteriormente en el *snippet* cómo realizar la petición, recibir la respuesta, extraer el cuerpo de la misma y transformarlo en un string. Para extraer la información del objeto JSON trabajando con Java necesitaremos el paquete `javax.json`.

Una posible forma de extraer la información requerida en el contexto en el que estamos trabajando, puede ser llevar a cabo los siguientes pasos:

- Crear un objeto `JsonReader` a partir del string que codifica dicho objeto JSON:

```
JsonReader lector = Json.createReader(new StringReader(respuesta))
```

- A partir de aquí hay que saber si el objeto JSON es propiamente un objeto o un array, actuando en consecuencia:

```
JsonObjet raiz = lector.readObject();
// O bien
JsonArray raiz = lector.readArray();
```

- Después habrá que aplicar los métodos:
  - `JsonObject getJsonObject(String name);`
  - `JsonArray getJsonArray(String name);`
  - `JsonNumber getJsonNumber(String name);`
  - `JsonString getJsonString(String name);`
  - `String getString(String name [, String defaultValue])`: atajo de `getJsonString(String name).getString();`
  - `int getInt(String name [, int defaultValue]);` atajo de `getJsonNumber(String name).intValue();`
  - `boolean getBoolean(String name [, boolean defaultValue]);`
  - ...

  a los objetos oportuna y convenientemente para ir obteniendo las distintas propiedades; si el objeto de referencia es un array JSON, entonces los métodos son similares, pero sustituyendo el parámetro `String name` por el entero correspondiente al índice. En cualquier caso esto requiere conocimiento a priori de la estructura (los parámetros entre corchetes denotan opcionalidad), para lo cual se sugiere que se estudie la documentación de `javax.json` y concretamente las interfaces `JsonObject`, `JsonArray`, `JsonValue`, ...

  Por ejemplo, si quisiéramos acceder al objeto `dia` tal y como aparece en la figura 12, entonces podríamos hacer lo como

```
JsonObject dia = raiz.getJsonArray(0).getJsonObject("prediccion").getJsonArray("dia");
```

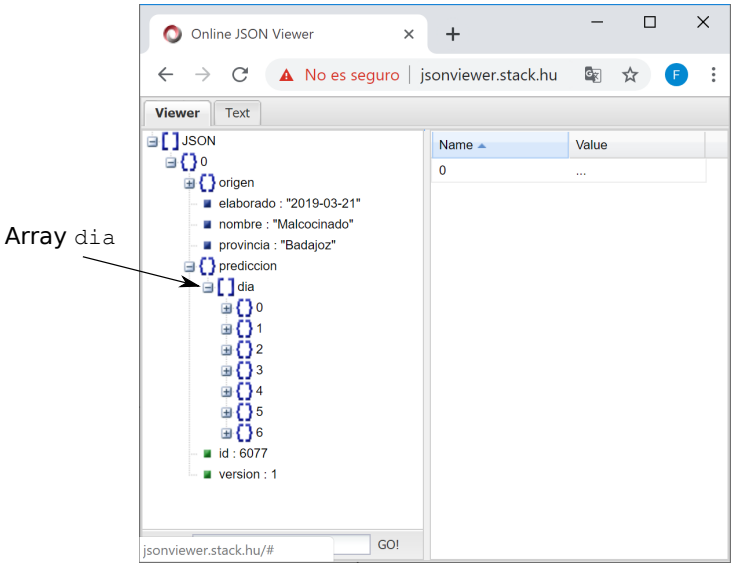

Figura 12: Visualización de una respuesta JSON

Obsérvese el ejemplo e intente entenderse, pues la forma de acceder a la información con esta perspectiva es propensa a ser confunsa. Tal y como se comentaba anteriormente, `dia` es un array; su posición 0-ésima (que es un objeto) se corresponde a la predicción del día actual, la 1-ésima, la del día siguiente, y así sucesivamente para toda la semana.
El siguiente *snippet* permite conocer la estructura del objeto JSON sin tener ningún conocimiento a priori del mismo:

```
public static void navigateTree(JsonValue tree, String key) {
   if (key != null)
      System.out.print("Key " + key + ": ");
   switch(tree.getValueType()) {
      case OBJECT:
         System.out.println("OBJECT");
         JsonObject object = (JsonObject) tree;
         for (String name : object.keySet())
            navigateTree(object.get(name), name);
         break;
      case ARRAY:
         System.out.println("ARRAY");
         JsonArray array = (JsonArray) tree;
         for (JsonValue val : array)
            navigateTree(val, null);
         break;
      case STRING:
         JsonString st = (JsonString) tree;
         System.out.println("STRING " + st.getString());
         break;
      case NUMBER:
         JsonNumber num = (JsonNumber) tree;
```

```
            System.out.println("NUMBER " + num.toString());
            break;
         case TRUE:
         case FALSE:
         case NULL:
            System.out.println(tree.getValueType().toString());
            break;
      }
   }
```

el cual podría ser empleado en nuestro ejemplo como

```
navegateTree(raiz,"");
```

Queremos insistir en que este *snippet* no es necesario para elaborar la práctica. Solo lo sería en el caso de desconocer a priori la estructura JSON; en nuestro caso, la podemos conocer a priori al visualizarla en un navegador o en el visor de JSON indicado.

4. Instalación de certificados digitales

Tal y como se habrá observado cuando se veían los contenidos de los objetos JSON en el navegador, el protocolo de acceso que aparecía en la barra de direcciones del navegador era obligatoriamente `https`, el cual emplea `TLS` (evolución de `SSL`). La conexión `https` con AEMET requiere que el servidor de AEMET envíe un certificado o cadena de certificados al navegador cliente para que éste compruebe que la información que se está recibiendo no ha sido alterada. La cadena de certificados debe acabar en un certificado autofirmado calificado de *certificado raíz de confianza* que el navegador ya debería tener instalado previamente en cierto *almacen de certificados*; en caso contrario, habrá que instalarlo explícitamente para que pueda funcionar correctamente y de forma segura. Si el certificado no está instalado, el navegador avisará de alguna manera que la conexión https no es segura. En el caso que nos ocupa, el certificado raíz de confianza se denomina ACCVRAIZ1 emitido por la organización ACCV (Agencia de Tecnología y Certificación Electrónica, `http:/www.accv.es`).

Esta serie de acciones llevadas a cabo por el navegador se deben trasladar al contexto de Java, ya que evidentemente no vamos a utilizar navegadores para conseguir esta información meteorológica, sino que todo lo programaremos en Java. Del mismo modo que con el navegador, hay que instalar el certificado raíz de la cadena de certificados enviados por el sitio web de AEMET como certificado de confianza, en algún sitio. Existen numerosas formas de conseguirlo, algunas más apropiadas que otras; en primera instancia, lo

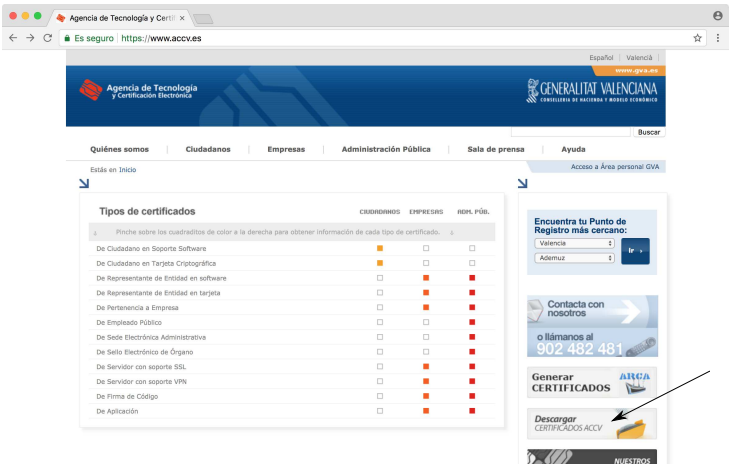

Figura 13: Página web de la agencia ACCV (`https://www.accv.es`).

podemos descargar desde la página web de ACCV; para ello, véanse las figuras 13 y 14. Se sugiere que el directorio destino sea `java\certs` desde la raíz del repositorio.

```
README.md
figuras
doc
java
    bin
    certs
        ACCVRAIZ1.crt
    paquetes
        javax.json-1.0.jar
        okhttp-3.10.0.jar
        okio-1.14.0.jar
    src
        test
            testAEMET.java
```

A continuación se sugiere que se visualice el certificado descargado. Ello es posible de manera directa simplemente haciendo doble click en el icono del fichero descargado (cuyo nombre muy probablemente será `ACCVRAIZ1.crt`), ya que Windows reconoce la extensión del mismo. Con ello podremos apreciar todas las características del mismo, e incluso verificar su validez a partir de la huella digital del mismo (debería coincidir con

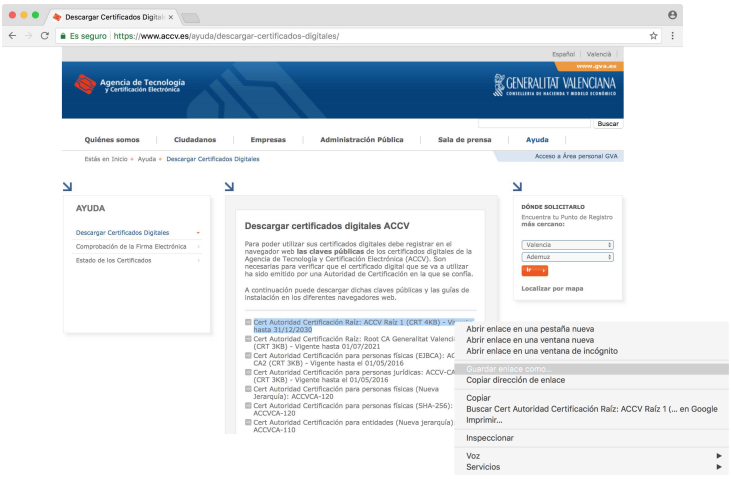

Figura 14: Descarga del certificado ACCVRAIZ1 de ACCV (fichero `ACCVRAIZ1.crt`).

`93057a8815c64fce882ffa9116522878bc536417` ;
en caso contrario habría que rechazarlo por erróneo o posible manipulación) tal y como puede apreciarse en la figura 15.

Java posee un almacen donde puede guardar los certificados de manera centralizada, pero también pueden emplearse almacenes *particulares* para tal efecto. Emplearemos esta última opción en la presente práctica. La ejecución del siguiente comando en línea de comandos

```
> keytool -import -file certs\ACCVRAIZ1.crt -storepass ESCRIBID_UNA_CLAVE_DE_ACCESO -keystore certs\NOMBRE_ALMACEN -alias ALIAS_DEL_CERTIFICADO
```

(obsérvese que `keytool` está ejecutado desde el directrio `java`.)

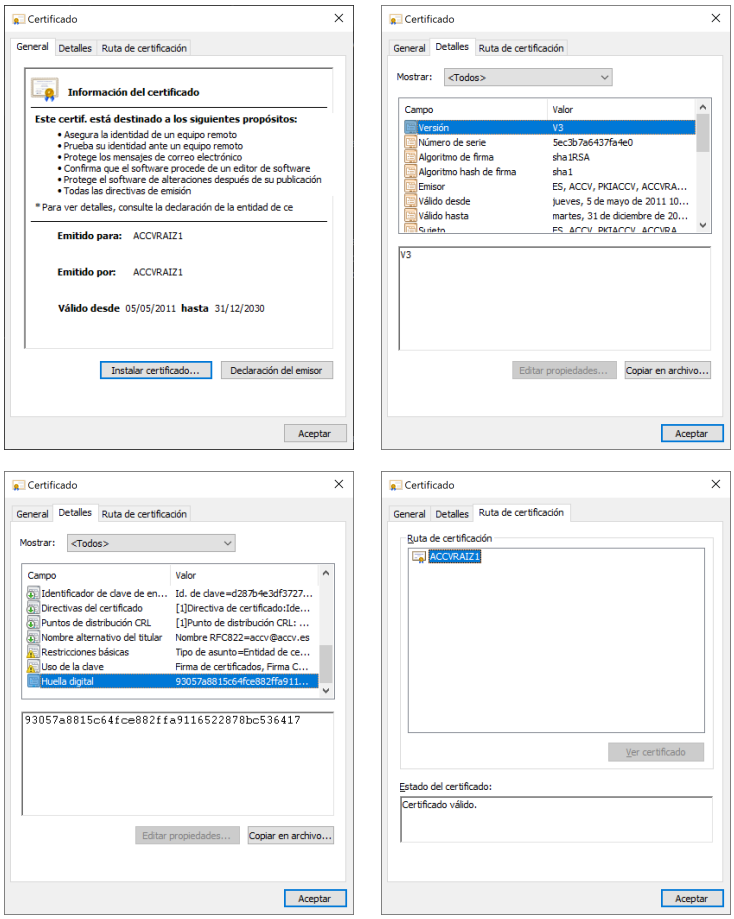

Figura 15: Visualización del fichero descargado `ACCVRAIZ1.crt`.

```
├── README.md
├── figuras
├── doc
├── java
│   ├── bin
│   ├── certs
│   │   ├── ACCVRAIZ1.crt
│   │   └── NOMBRE_ALMACEN
│   ├── paquetes
│   │   ├── javax.json-1.0.jar
│   │   ├── okhttp-3.10.0.jar
│   │   └── okio-1.14.0.jar
│   └── src
│       └── test
│           └── testAEMET.java
```

consigue guardar el certificado digital contenido en el fichero `ACCVRAIZ1.crt` (en principio ese será muy probablemente el nombre de dicho fichero) en un almacen de claves (en un fichero cuyo contenido serán certificados digitales y claves en general) cuyo nombre (`NOMBRE_ALMACEN` en el ejemplo, aunque póngase el que se desee —luego será necesario recordarlo en el código Java—) es el que acompaña al parámetro `-keystore`, donde

- `ESCRIBID_UNA_CLAVE_DE_ACCESO`: puede ser cualquier clave (no será necesaria en Java si sólo leemos certificados)
- `NOMBRE_ALMACEN`: nombre del fichero que almacenará claves y certificados.
- `ALIAS_DEL_CERTIFICADO`: forma de identificarlo fácilmente dentro del almacén.

El comando `keytool` viene con la distribución del JDK que esté descargada en el ordenador, por lo que tiene las mismas consideraciones de ejecución que, por ejemplo, `javac`.

Tendremos que indicar a la máquina virtual dónde puede encontrar el certificado que le hará falta para validar la conexión https. Para ello, podemos emplear la siguiente instrucción en nuestro código Java antes de proceder a realizar cualquier petición `HTTPS` hacia el servidor:

```
System.setProperty("javax.net.ssl.trustStore", "certs\\NOMBRE_ALMACEN");
```

Recuérdese que aquí `NOMBRE_ALMACEN` (que debería ser un `String`) denota el nombre del fichero que contiene el almacén de certificados generado anteriomente mediante la herramienta `keytool`, añadiendo el *path* oportuno (`certs\`).

Otra reseña importante es que de manera implícita se está suponiendo que nuestro código Java se ejecutará desde el directorio `java`, ya que desde ahí se estará accediendo al almacén de la forma indicada `"certs\\NOMBRE_ALMACEN"`, tal y como se apreciaría en el árbol

```
├── README.md
├── figuras
├── doc
└── java
    ├── bin
    ├── certs
    │   ├── ACCVRAIZ1.crt
    │   └── NOMBRE_ALMACEN
    ├── paquetes
    │   ├── javax.json-1.0.jar
    │   ├── okhttp-3.10.0.jar
    │   └── okio-1.14.0.jar
    └── src
        └── test
            └── testAEMET.java
```

Otro detalle relativamente importante es la forma con la que se ha especificado el separador de directorios, que en Windows adopta la forma \. Se especifica con un carácter doble, porque si fuera simple, dentro un *string* se interpreta como un carácter de *escape*, dando resultados erróneos; de esta forma, en un *string*, `"\\"` se interpreta como un \. En Linux o OS-X, no tiene trascendencia, ya que el separador es `/`, el cual se interpreta como tal dentro de un *string*. En cualquier caso, Java ejecutado en Windows también puede interpretar el carácter `/` como separador de directorios.

5. Desarrollo de la práctica

Una vez familiarizados con el funcionamiento de las API de AEMET OpenData, e instalado el certificado raíz de ACCV, a continuación se propone averiguar cuál será la predicción de los parámetros meteorológicos que a continuación se enumeran:

- Temperatura (a las 18 h —intervalo 12–18 h—)
- Probabilidad de precipitación en tanto por cien entre las 12 y las 18 h
- Dirección y velocidad del viento entre las 12 y las 18 h
- Estado del cielo entre las 12 y las 18 h

del día siguiente al que se realice la práctica, en alguna de los siguientes poblaciones de España:

- Peleas de Abajo, provincia de Zamora.
- Guarromán, provincia de Jaén
- Malcocinado, provincia de Badajoz

Deben contrastarse los resultados con los obtenidos visualmente de manera directa de la página web de AEMET.

Indicaciones generales:

- Clónese el repositorio localmente, consiguiendo la estructura comentada anteriormente:

```
├── README.md
├── figuras
├── doc
└── java
    ├── bin
    ├── certs
    ├── paquetes
    │   ├── javax.json-1.0.jar
    │   ├── okhttp-3.10.0.jar
    │   └── okio-1.14.0.jar
    └── src
        └── test
            └── testAEMET.java
```

- Instálese el certificado de ACCV en el directorio `java\certs` y créese el almacen de claves en el mismo directorio.
- Respétese la condición ya indicado en el fichero oportuno de que la clase `testeAEMET` pertenezca al paquete `test`.

- Identifíquese los paquetes necesarios (`okhttp3`, `json`, ...) e impórtense las clases de la manera adecuada sintácticamente. (recuérdese que con `jar tvf fichero.jar` se podía visualizar el contenido de un fichero `jar`).
- Realícense pruebas sencillas de complejidad creciente:
  - Diséñese la API REST (URL con la petición) con la que se realizará la petición.
  - Obsérvese el resultado en un navegador
  - Procédase entonces en Java con el *snippet* propuesto.
    - Cácense las excepciones oportunamente.
    - Diséñese la línea de compilación con `javac` con los *switches* necesarios para que la compilación funcione correctamente en *línea de comandos*. Déjense los ficheros `.class` resultantes de la compilación en la carpeta `bin`.
    - Diséñese la línea de ejecución con `java` con los *switches* necesarios para que la ejecución en línea de comandos funcione correctamente en *línea de comandos*.
    - Compílese y ejecútese desde el directorio de trabajo `java`
  - Repítanse los pasos anteriores con la segunda petición
  - Visualícense las respuestas con un visor JSON
  - Vayánse escribiendo poco a poco resultados parciales hasta conseguir los definitivos.

El profesorado realizará la siguiente prueba para verificar el correcto funcionamiento:

- Clonará el repositorio
- Cambiará al directorio de trabajo `java`
- Efectuará la compilación

```
javac -cp ... -d ... src/test/testAEMET.java
```

- Si el resultado es correcto, entonces:

```
java -cp ... test.testAEMET
```

Se sugiere, por tanto, que se intenten imitar estos pasos para saber a priori, si la calificación será correcta o no.

6. Resultados a entregar

Al finalizar la sesión de prácticas, debe actualzarse de manera inmediata el repositorio en GitHub, indicando mediante un *tag* la denominación **pr2.0** (recuérdese subir los *tags* explícitamente, ya que no es suficiente con actualizar el repositorio remoto con el local). Se dispondrá de una semana para finalizar detalles que queden pendientes. Las sucesivas versiones operativas que se suban a GitHub deben estar etiquetados con **pr2.1**, **pr2.2**, etc.

Importante: Muéstrese al profesor la evolución de todos los apartados de la práctica.