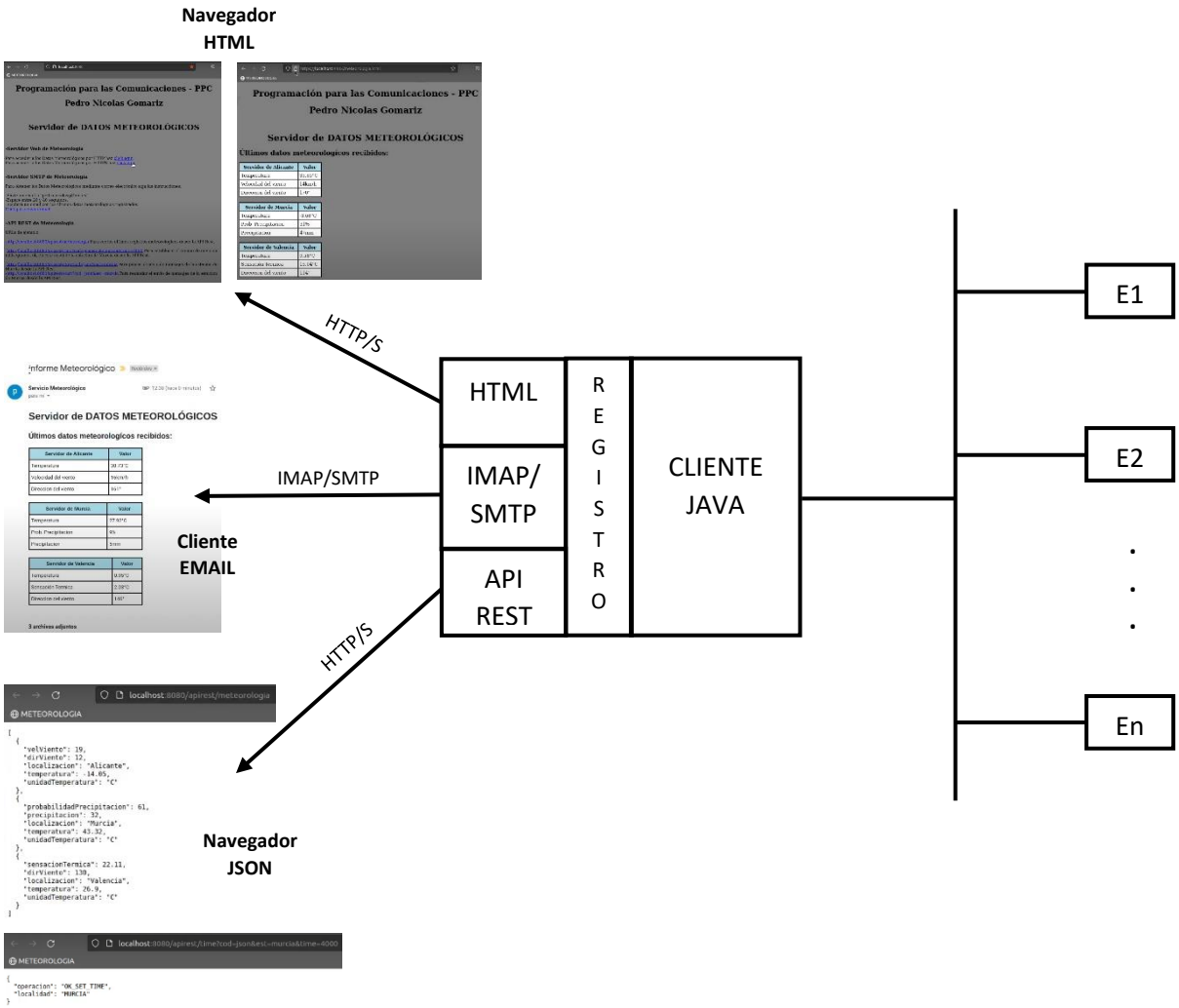
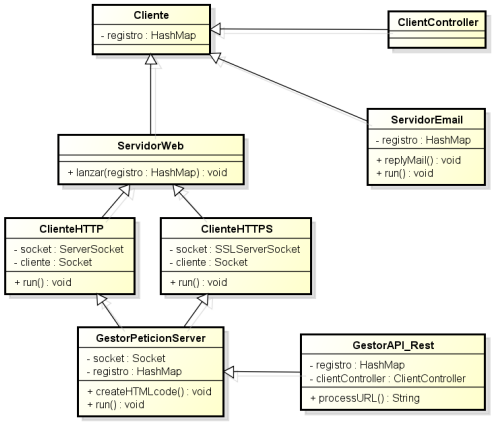
Programación para las Comunicaciones

Práctica 3
HTTP/S - SMTP – REST



Pedro Nicolas Gomariz
pedro.nicolasg@um.es
48753907W

1.- Descripción sobre el diseño e implementación y listado de clases incluyendo los diagramas de clases más relevantes.



Para la realización de esta práctica partimos del código, y por tanto de las clases ya existentes y explicadas en las prácticas anteriores.

Por un lado, partiendo del código de la pract. 2, añadimos un nuevo atributo de tipo HashMap en la clase *Cliente*:

```
HashMap<String, DatosEstacion> registro;
```

En este atributo estaremos almacenando para cada estación de la cual recibamos un mensaje de difusión, por un lado su nombre como *key* del mapa y por otro lado un objeto de la clase *DatosEstacion* con los últimos datos meteorológicos recibidos, esto lo estaremos almacenado como *value* del mapa.

Por otro lado cojeremos el código de la práctica 1 y crearemos un objeto de tipo *ServidorWeb* en la clase *Cliente*, aquí realizaremos ciertas modificaciones que explicaremos en el apartado 3.

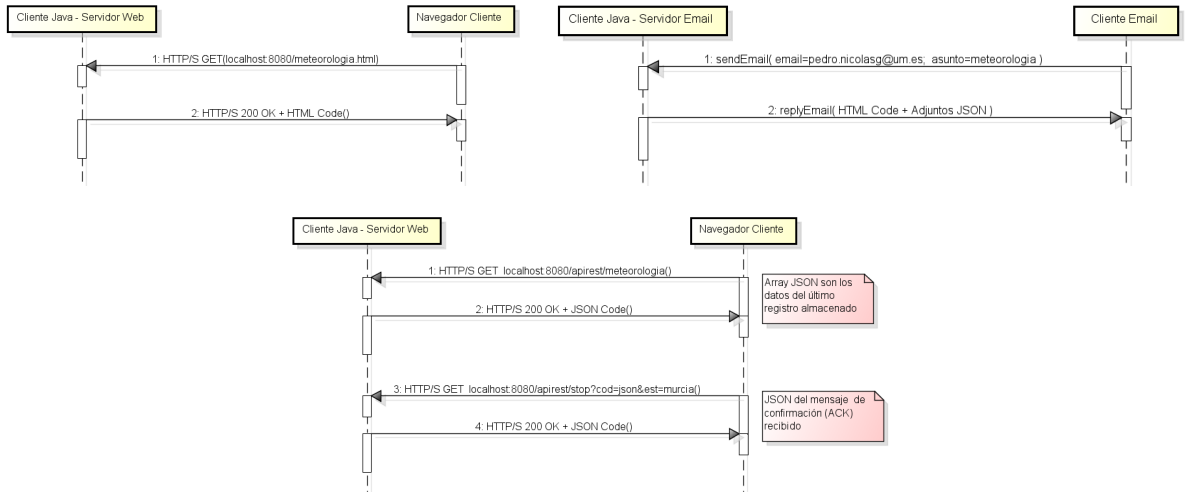
Posteriormente crearemos una nueva clase *ServidorEmail*, la cual será la encargada de realizar la funcionalidad correspondiente al servicio IMAP/SMTP, más adelante en el apartado 4 explicaremos los detalles.

Por último, para la gestión de la API Rest crearemos una nueva clase la cual será utilizada por la clase *GestionPeticionServer* para procesar las distintas URLs que nos puedan llegar relacionadas con la misma. Esto lo analizaremos en el apartado 5.

Todos estos servicios tendrán acceso a la variable *registro* para obtener los últimos datos recibidos y poder construir así las respuestas a las peticiones que le generen sus clientes.

Esta variable será actualizada cada vez que se reciba un nuevo mensaje de difusión, en el caso de que ya haya una entrada en el mapa cuya *key* coincida, el *value* será reemplazado por el nuevo, en el caso de que no exista se añadirá una nueva entrada al mapa.

2.- Descripción del protocolo mediante diagramas de secuencia.



Los diagramas de secuencia de esta práctica resultan sencillos, puesto que se trata de protocolos sin estado en los cuales hay una petición y asociado a ella una respuesta, y las consecutivas peticiones no tienen relación alguna con las anteriores.

En el caso de HTTP/S el usuario realiza una consulta a una URL a través del navegador y en función del recurso solicitado el servidor web responderá. En el apartado 3 y 5 definiremos dicho comportamiento.

Para el servicio de API Rest también se hace uso del servidor web mencionado anteriormente para gestionar las distintas URLs relacionadas con el servicio Rest, las cuales implementan las mismas funcionalidades de control que se implementaron en la práctica 2 mediante la línea de comandos de la terminal de control de cliente.

Por último, para el servicio de EMAIL, únicamente deberá enviar un correo a la dirección pedro.nicolasg@um.es utilizando "*meteorologia*" como asunto. Tras esto y esperar entre 15 y 30 segundos, el usuario recibirá un email con la última información recibida por cada estación. Los detalles de esto los analizaremos en el apartado 4.

3.- Implementación Servidor Web.

Para implementar el servicio HTTP/S, se parte del código de la práctica 1. Puesto que la práctica se realizó de forma muy modularizada no se ha tenido que modificar nada relacionado con sockets, certificados, etc, únicamente se ha tenido que modificar la clase *GestorPeticionServer* de la cual se ha eliminado la funcionalidad que se implementó para la práctica 1 y se ha añadido distintos *if-else* para comprobar los recursos que nos están solicitando y actuar en consecuencia devolviendo lo oportuno.

En el caso de que se soliciten los recursos http://localhost:8080/ ó http://localhost:8080/index.html devolveremos un HTTP 200 OK junto a un código HTML, mostrándole al usuario una página web de acceso múltiple donde podrá observar el funcionamiento de los tres servicios y algunas URLs de ejemplo, relacionadas con la API Rest y que comentaremos más adelante.

En el caso que se solicite el recurso http://localhost:8080/meteorologia.html devolveremos un HTTP 200 OK y además un código HTML con una representación de los últimos datos de las estaciones registradas.

En el caso de recibir un enlace del tipo http://localhost:8080/apirest/* entonces crearemos un objeto de la clase *GestionAPI_Rest* la cual explicaremos más adelante y se encargará de gestionar la URL.

Por último, si se solicita algún recurso distinto a los mencionados anteriormente se devolverá un HTTP 404 junto a un código HTML que le indicará al usuario el mensaje de error.

4.- Implementación Servidor de Email.

Para implementar el servicio de EMAIL he creado la clase *ServidorEmail* la cual se instancia desde la clase *Servidor*. Esta nueva clase correrá en un nuevo hilo el cual se encargará de cada 15 segundos ejecutar una función. Esta función se denomina *replyMail()* y su función es acceder al email de pedro.nicolasg@um.es mediante el protocolo de acceso a correo IMAP y obtener una lista de aquellos mensajes que tiene el flag *SEEN* a false, es decir, los mensajes no leídos, posteriormente recorreremos esta lista de mensajes y únicamente nos quedaremos con aquellos que tengan como asunto "*meteorologia*". Por ultimo, cojeremos cada uno de estos mensajes y responderemos a ellos mediante un mensaje de tipo MultiPart, donde una parte será el cuerpo del mensaje, en HTML para que el cliente lo pueda observar de forma "amigable" los datos, además tendrá n partes más, una por cada estación, estas partes serán archivos adjuntos que tendrá el email de respuesta y contendrá el

código JSON de la estación correspondiente con los últimos datos meteorológicos, estos se obtienen del *HashMap registro* que se actualiza en la clase *Cliente* y que se comparte con *ServidorEmail*. El mensaje de respuesta se enviará al cliente mediante el protocolo de email SMTP. Se podrá el flag *SEEN* a true para el mensaje que se recibió del cliente, de forma que evitaremos volverlo a tratar.

5.- Implementación API Rest.

Para implementar el servicio REST, se crea la clase *GestorAPI_Rest*, la cual se encargará de gestionar las URLs que comiencen por http://localhost:8080/apirest/*. Para esto se apoya en una función denominada *processURL()* la cual retorna un *String* con la respuesta que debe devolver el servidor web y tendrá como parámetro la propia URL a gestionar.

En el caso de la que URL sea http://localhost:8080/apirest/meteorologia entonces será una petición relacionada con la parte de difusión y se estará solicitando los últimos datos recibidos por las estaciones, por lo que como esta clase tiene acceso a la variable *registro* ya comentada anteriormente puede crear la respuesta incluyendo un contenido de tipo "*application/JSON*" y como contenido un array JSON con los últimos valores meteorológicos recibidos por parte de las estaciones.

Para el caso de la parte de control se implementan otras distintas URLs como las que se adjuntan:

- http://localhost:8080/apirest/time?cod=json&est=murcia&time=4000
- http://localhost:8080/apirest/stop?cod=json&est=murcia
- http://localhost:8080/apirest/start?cod=json&est=murcia
- http://localhost:8080/apirest/codeJSON?cod=json&est=murcia
- http://localhost:8080/apirest/codeXML?cod=json&est=murcia
- http://localhost:8080/apirest/tempC?cod=json&est=murcia
- http://localhost:8080/apirest/tempK?cod=json&est=murcia
- http://localhost:8080/apirest/tempF?cod=json&est=murcia

Como se puede observar en las URLs en primer lugar se indica la acción a realizar y después mediante la variable *"cod"* que debe ser *JSON o XML* se elige la codificación en la que se quiere enviar el mensaje de control a la estación, mediante la variable *"est"* se indica el nombre de la estación en la que se quiere realizar la acción, y en el caso de la URL de *time*, mediante la variable *"time"* se indica el tiempo en milisegundos para indicarle a la estación cada cuanto queremos recibir un mensaje suyo de difusión.

En el caso de que se introduzca una URL no válida, mal formada, con algún error en las variables, o indicando en la variable *"est"* el nombre de una estación que no existe, o en el caso de que hayan pérdidas y una estación no responda al mensaje de control, entonces se devolverá un objeto JSON indicando que se ha producido un error.

Para la implementación de las distintas URLs de control, lo que se ha hecho es que la clase *GestorAPI_Rest* tenga acceso a la clase *ClientController* de la clase *Cliente*, por lo que se podrá hacer uso del método *processComand()* de dicha clase, el cual se encarga de procesar los comandos de control introducidos en la consola de control de cliente. Esto nos servirá puesto que las URLs Rest relacionadas con la parte de control serán traducidas en comandos que tienen el mismo efecto que el esperado por la URL Rest, por ejemplo http://localhost:8080/apirest/tempK?cod=json&est=murcia se traducirá en el comando de consola de cliente java "*json murcia ut k*" y de igual forma para el resto de las URLs con el resto de comandos que ya explicamos en la práctica 2. Por lo que realizándose de esta forma estaremos reutilizando toda la funcionalidad (tratamiento de peticiones, envío de mensajes de control, mecanismo de pérdidas de mensajes mediante temporizaciones y ACKs, recepción de mensajes de confirmación, etc…) ya implementada anteriormente.