ESCUELA DE INGENIERÍA DE BILBAO
EXAMEN ORDINARIO DE GRADO – 25/05/2022
ASIGNATURA: Sistemas Web
GRADO: Ingeniería Informática de Gestión y Sistemas de Información

Nombre y apellidos:

Notas:

- *Duración examen: 75 min.*
- *Algunas preguntas de test pueden tener **más de una** respuesta correcta. Para que la pregunta sea considerada correcta se deben marcar únicamente todas las respuestas correctas.*

1. **(0,5)** ¿En *OAuth 2.0*, a quién identifican los parámetros *client_key* y *client_secret*?

a. Al usuario de la aplicación.
b. Al programador de la aplicación
c. A la aplicación.
d. Al cliente que ha encargado la aplicación.
e. Ninguna de las anteriores.

2. **(0,5)** El objeto **XMLHttpRequest**…

a. Permite codificar una petición HTTP en XML.
b. Se puede utilizar en la cabecera Content-Type para indicar que el formato del contenido de la petición es XML.
c. Se utiliza en los métodos *doGet()* y *doPost()* de un servlet para referenciar la petición HTTP.
d. Permite realizar peticiones HTTP desde código javascript.
e. Permite enviar el contenido de una petición HTTP en XML.

3. **(0,5)** Un servidor recibe la siguiente petición http:

```
GET /html/Sistemas Web.html HTTP/1.1
Host: sw2022.ehu.eus
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

La respuesta del servidor será (justifica la respuesta):

a. 200 - OK, en el cuerpo del mensaje se enviará la página html.
b. 400 - Bad Request, petición incorrecta.
c. 404 - File Not Found, el servidor no ha encontrado la página.
d. 405 - Method Not Allowed, el método GET no implementado en servidor.

4. **(1)** A continuación, se muestra parte del código de un JSP.

```
<%@ page import="java.util.*"%>
<% int i=0; %>
...
<body>
<% for(i=1; i<3; i++) { %>
Server Date: <%= new Date().toString() %><br/>
<% } %></br>
<script language="javascript">
for(i=2; i<<%= i %>; i++) {
var data = new Date();
document.write("Client Date: ");
document.write(data);
document.write("<br/>");
}
</script></body>
```

Suponiendo que el JSP se procesa en el servidor en la fecha ***Fri, 21-May-2022 08:33:47 GMT*** y se recibe en el navegador en la fecha ***Fri, 21-May-2022 08:33:50 GMT***, escribe el contenido que se mostrará en el navegador.

Respuesta:
Server Date: Fri, 21-May-2018 08:33:47 GMT
Server Date: Fri, 21-May-2018 08:33:47 GMT
Client Date: Fri, 21-May-2018 08:33:50 GMT

5. **(1)** Describe detalladamente la estructura de una petición HTTP y pon un ejemplo.

Respuesta:

FUNCIONAMIENTO DE HTTP:

SOLICITUD DEL CLIENTE

Sintaxis de una solicitud HTTP

Método URI HTTP/1.1
Cabeceras
CRLF
Cuerpo del mensaje

solicitud HTTP del ejemplo

GET /recurso HTTP/1.1
Host: sw2021.com:8080
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Desktop

Método: GET
El método describe el tipo de acción CRUD (Create, Read, Update and Delete) que se desea llevar a cabo sobre el recurso. En este caso, GET → Lectura.

URI: /recurso
La identificación del recurso puede realizarse con la URI completa o la URI relativa.

GET http://sw2021.com:8080/recurso HTTP/1.1

GET /recurso HTTP/1.1
Host: sw2021.com:8080

Cabeceras: caracterizan determinados aspectos del cliente y indican preferencias sobre la respuesta.
Accept: el navegador indica que acepta contenido en HTML.
Accept-Encoding: el navegador indica que prefiere contenido comprimido (en formato gzip), aunque también acepta contenido no comprimido (identity)
Accept-Language: el navegador indica que su preferencia de idioma es el inglés y su segunda opción el castellano.
User-Agent: el navegador se identifica como Mozilla sobre una plataforma Windows de escritorio.

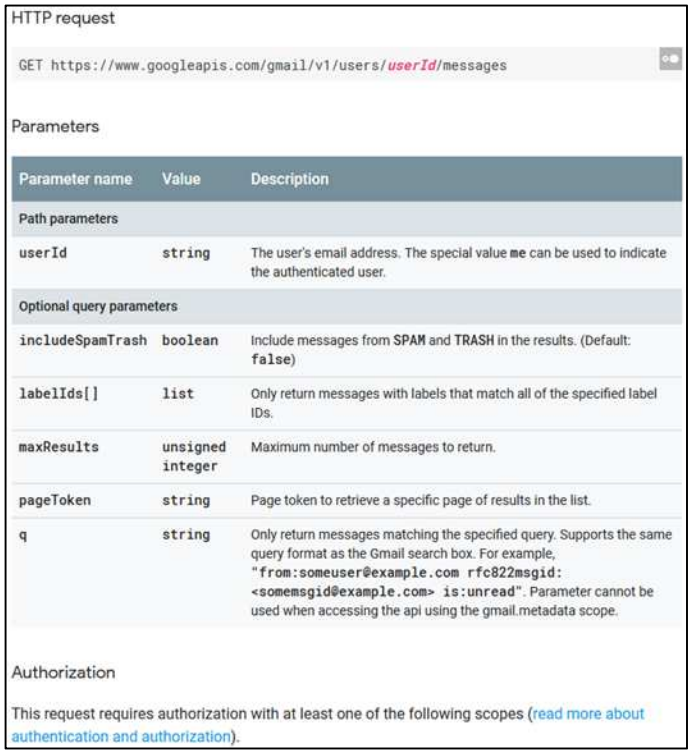
Cuerpo del mensaje: en este caso está vacío.

Oskar Casquero y María Luz Álvarez
Dpto. Ingeniería de Sistemas y Automática
2

6. **(1)** En la aplicación web *miApp*, se desea asignar la URI http://ehu.eus/miApp/DoLogin/NewServlet al servlet *NewServlet* que se encuentra en el paquete *eus.ehu*. Completa el fichero de despliegue.

```
<web-app>
    <servlet>
       <servlet-name> NewServlet <servlet-name>
       <servlet-class> eus.ehu.NewServlet </servlet-class>
    </servlet>
    <servlet-mapping>
       <servlet-name> NewServlet</servlet>
       <url-pattern> /DoLogin/NewServlet <url-pattern>
    <servlet-mapping>
</web-app>
```

7. **(1,5)** Con la ayuda de la documentación adjunta del API de Gmail, escribe la petición **HTTP** que permite a un usuario autenticado recuperar los mensajes no leídos con remitente sw2022@ehu.es. NOTA: La petición no debe devolver código 400.



HTTP request

`GET https://www.googleapis.com/gmail/v1/users/userId/messages`

Parameters

Parameter name	Value	Description
Path parameters		
`userId`	`string`	The user's email address. The special value **me** can be used to indicate the authenticated user.
Optional query parameters		
`includeSpamTrash`	`boolean`	Include messages from **SPAM** and **TRASH** in the results. (Default: `false`)
`labelIds[]`	`list`	Only return messages with labels that match all of the specified label IDs.
`maxResults`	`unsigned integer`	Maximum number of messages to return.
`pageToken`	`string`	Page token to retrieve a specific page of results in the list.
`q`	`string`	Only return messages matching the specified query. Supports the same query format as the Gmail search box. For example, `"from:someuser@example.com rfc822msgid: <somemsgid@example.com> is:unread"`. Parameter cannot be used when accessing the api using the gmail.metadata scope.

Authorization

This request requires authorization with at least one of the following scopes (read more about authentication and authorization).

Respuesta:

```
GET  /gmail/v1/users/me/messages?q=sw2019%40gmail.com+is%3Aunread  HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer [YOUR_ACCESS_TOKEN]
```

8. **(1,5)** Para que una aplicación cree un evento en la cuenta de Google Calendar de un usuario, utilizamos el siguiente código.

```
scope = 'https://www.googleapis.com/auth/calendar'
calendarioID = 'addressbook#contacts@group.v.calendar.google.com'
cabeceras = {}
cabeceras['User-Agent'] = 'Python Client'
# suponer que access_token se ha definido y obtenido correctamente antes
cabeceras['Authorization'] = 'Bearer ' + access_token
cabeceras['Content-Type'] = 'application/json'
url = 'https://www.googleapis.com/calendar/v3/calendars/'+calendarioID+'/events'
cuerpo = { 'end'         : { 'date': '2022-05-25' },
           'start'       : { 'date': '2022-05-24' },
           'description': 'Sistemas Web',
           'summary'     : 'EXAMEN' }
cuerpo = json.dumps(cuerpo)
respuesta = requests.post(url, headers=cabeceras, data=cuerpo)
print(respuesta.status_code)
print(respuesta.content)
```

Al ejecutar la aplicación, nos devuelve el error **404 Not Found**. Las causas de esta respuesta pueden ser:

- Hay que indicar la longitud de los datos en la cabecera *Content-Length*.
- El cuerpo del mensaje no está correctamente formateado.
- El usuario no tiene un calendario con ese identificador.
- No se ha definido correctamente la cabecera *Content-Type.*
- Se ha definido mal el *scope* de la aplicación.

Tacha lo que no proceda y justifica por qué pueden ser validas o no cada una de las opciones.

Respuesta:
1.- INCORRECTA: Con el método POST es necesario definir la cabecera Content-Length, pero el método requests.post() la rellena automáticamente.

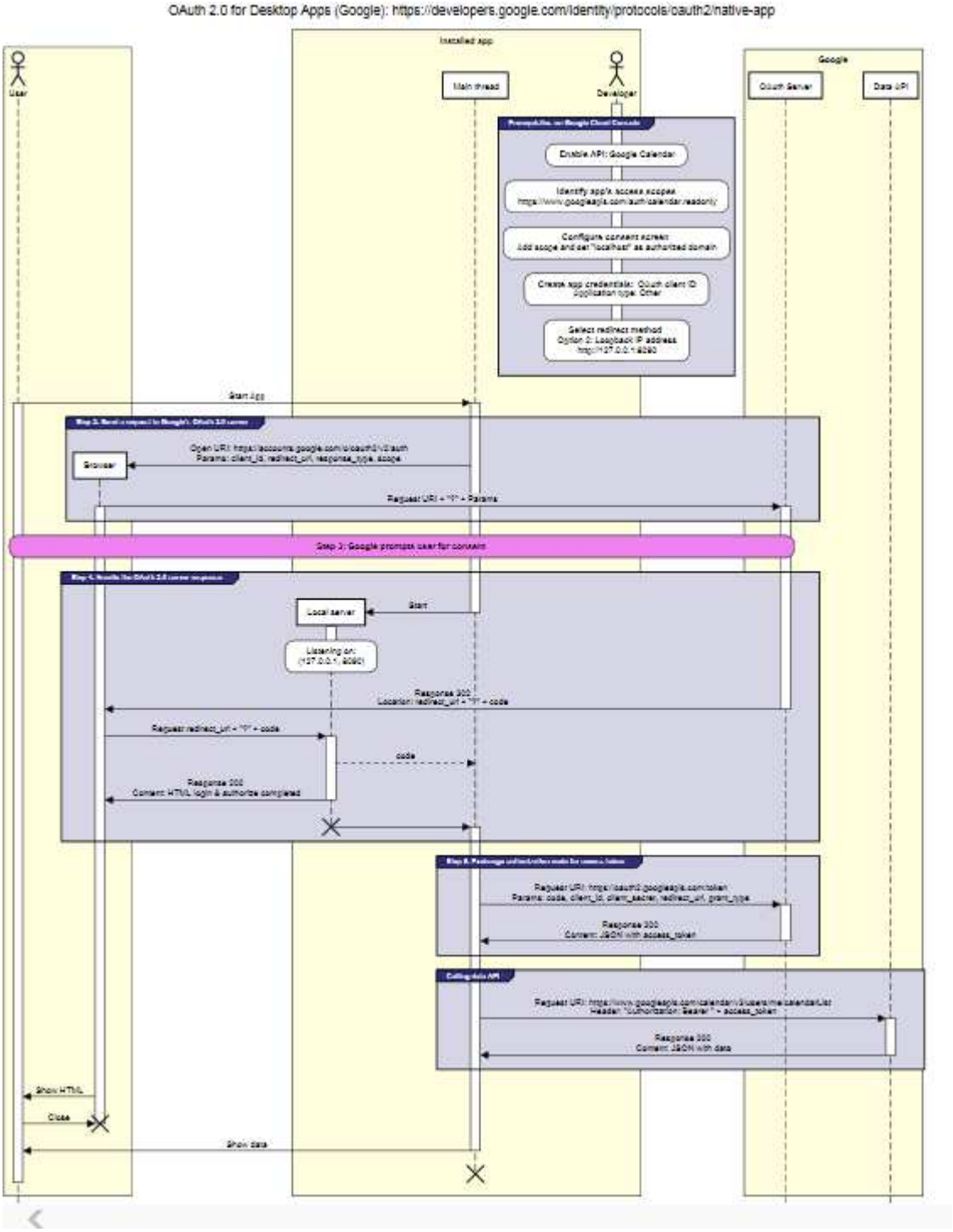
2.- INCORRECTA: El cuerpo del mensaje se formatea como json en el código.
cuerpo = json.dumps(cuerpo)
En caso de no formatear, el mensaje de error sería 400 Bad Request.

3.- CORRECTA: Esta es la causa del error. Como el id del calendario aparece en la identificación del recurso, si el calendario no existe, el servidor no puede encontrar el recurso.
url = 'https://www.googleapis.com/calendar/v3/calendars/'+calendarioID+'/events'

4.- INCORRECTA: La cabecera si está definida en el código:
cabeceras['Content-Type'] = 'application/json'

5.- INCORRECTA: El scope es correcto y permite accede a todo el contenido de calendar
scope = 'https://www.googleapis.com/auth/calendar'
Si el scope no es correcto, el error es: 403 Forbidden.

9. **(1,5)** Completa el siguiente diagrama de secuencia.



10. (1) Un estudiante programa la siguiente petición HTTP para responder a un tweet cuyo ID es 866344856981041155 y que tiene como autor al usuario de Twitter *ocasquero* (es decir, enviar un tweet en respuesta a otro tweet).

```
POST https://api.twitter.com/1.1/statuses/update.json?status=Hola%2C+%40ocasquero&in_reply_to_status_id=866344856981041155
Host: api.twitter.com
User-Agent: Google App Engine
Authorization: OAuth oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
                     oauth_nonce="a9900fe68e2573b27a37f10fbad6a755",
                     oauth_signature="39cipBtIOHEEnybAR4sATQTpl2I%3D",
                     oauth_signature_method="HMAC-SHA1",
                     oauth_timestamp="1318467427",
                     oauth_token="NPcudxy0yU5T3tBzho7iCotZ3cnetKwcTIRlX0iwRl0",
                     oauth_version="1.0"
```

En respuesta a dicha petición, se recibe un código 403. A partir de la documentación adjunta, indica cuáles de las siguientes afirmaciones son correctas:

a. En dicha petición se ha enviado un tweet cuyo contenido ya ha sido publicado previamente.
b. En dicha petición se ha enviado un tweet cuyo contenido comienza por “D” o “M”.
c. En dicha petición no se ha indicado el parámetro opcional *media_ids*.
d. Con dicha petición se ha sobrepasado el número máximo de tweets que un usuario puede publicar a través del API en un intervalo de tiempo dado.

POST statuses/update

Updates the authenticating user’s current status, also known as Tweeting.

For each update attempt, the update text is compared with the authenticating user’s recent Tweets. Any attempt that would result in duplication will be blocked, resulting in a 403 error. A user cannot submit the same status twice in a row.

While not rate limited by the API, a user is limited in the number of Tweets they can create at a time. If the number of updates posted by the user reaches the current allowed limit this method will return an HTTP 403 error.

Resource URL

https://api.twitter.com/1.1/statuses/update.json

Resource Information

Response formats	JSON
Requires authentication?	Yes (user context only)

Parameters

Name	Required	Description
status	required	The text of the status update. URL encode is necessary. t.co link wrapping may affect character counts. There are some special commands in this field to be aware of. For instance, preceding a message with “D” or “M” and following it with a screen name can create a Direct Message to that user if the relationship allows for it.
in_reply_to_status_id	optional	The ID of an existing status that the update is in reply to. **Note:** This parameter will be ignored unless the author of the Tweet this parameter references is mentioned within the status text. Therefore, you must include *@username* , where *username* is the author of the referenced Tweet, within the update.
media_ids	optional	A list of *media_ids* to associate with the Tweet. You may include up to 4 photos or 1 animated GIF or 1 video in a Tweet. See Uploading Media for further details on uploading media.