ESCUELA DE INGENIERÍA DE BILBAO
EXAMEN ORDINARIO DE GRADO – 26/05/2023
ASIGNATURA: Sistemas Web
GRADO: Ingeniería Informática de Gestión y Sistemas de Información

Nombre y apellidos:

Notas:

- *Duración examen: 90 min.*
- *En cada pregunta aparece la puntuación correspondiente.*

1. (1 punto) Un desarrollador desea realizar una petición HTTP para descargar un código QR mediante sockets, ya que el microcontrolador sobre el que está desarrollando la aplicación no dispone de los recursos necesarios para instalar y utilizar una librería HTTP. Para ello, el desarrollador debe rellenar esta plantilla de código:

```
HOST = "alias.com"
PORT = "80"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
print("Local socket:", s.getsockname())

metodo = "GET"
recurso = "/img/codigo_qr.png"
primera_linea = metodo + " " + recurso + " HTTP/1.1\r\n"

cabeceras = { 'Host': HOST }
cadena_cabeceras =''
for each in cabeceras:
    cadena_cabeceras = each + ":" + cabeceras[each] + "\r\n"

cuerpo = ''

cadena_peticion_http = primera_linea + cadena_cabeceras + "\r\n"

cadena_peticion_http_bytes = bytes(cadena_peticion_http, 'utf-8')

print('\n##### HTTP eskaera #####')
print(cadena_peticion_http_bytes)
s.sendall(cadena_peticion_http_bytes)

data = s.recv(1024)
print('\n##### Respuesta HTTP #####')
print(data)

s.close()
```

Rellena la plantilla de código haciendo uso de mínimo número de cabeceras y suponiendo que la URI de la imagen es http://alias.com/img/codigo_qr.png

2. (0,5 puntos) A continuación, se muestra un segmento de código en Python que forma parte de un programa más amplio. Este segmento se ha programado para realizar la identificación de usuario en eGela. NOTAS:

- Las variables *cookiea* y *logintoken* contienen la cookie y el token de autenticación obtenidos previamente dentro del mismo programa.
- Las variables *USERNAME* y *PASSWORD* contienen el identificador y la contraseña introducidos por el usuario por línea de comandos.

```
metodo = 'POST'
uri = "https://egela.ehu.eus/login/index.php"
cabeceras = {'Host': 'egela.ehu.eus',
             'Cookie': cookiea,
             'Content-Type': 'application/x-www-form-urlencoded',}
datos = 'username=' + USERNAME + \
        '&password=' + PASSWORD + \
        '&logintoken' + logintoken }
cabeceras['Content-Length'] = str(len(datos))
respuesta = requests.request(metodo, uri, headers=cabeceras, \
                             data=datos, allow_redirects=False)
```

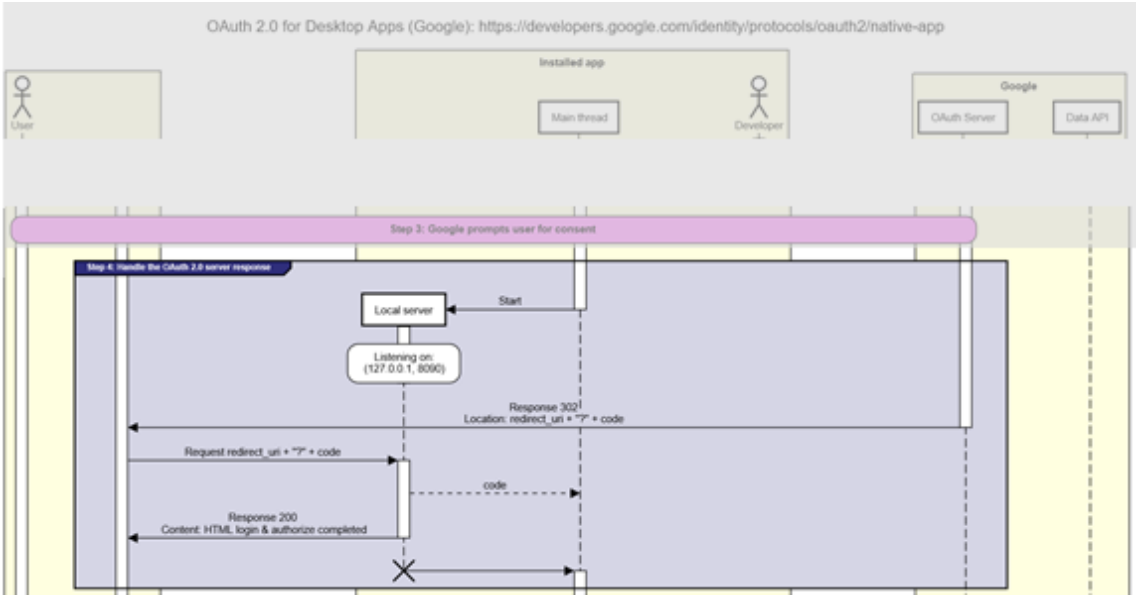
Tres usuarios diferentes usan el programa: dos de ellos consiguen identificarse correctamente, mientras que el tercero que no lo consigue. Suponiendo que la cookie y el token de autenticación se obtienen correctamente en cada intento de inicio de sesión, y que todos los usuarios introducen correctamente sus credenciales, ¿a qué se debe el error de identificación del tercer usuario? ¿Cómo se puede solucionar?

Los datos no se han codificado como código URL/URI (también llamado código *por ciento*). Por tanto, si el tercer usuario tiene algún carácter especial en sus credenciales, éste no se codificará correctamente para su envío por HTTP. Solución:

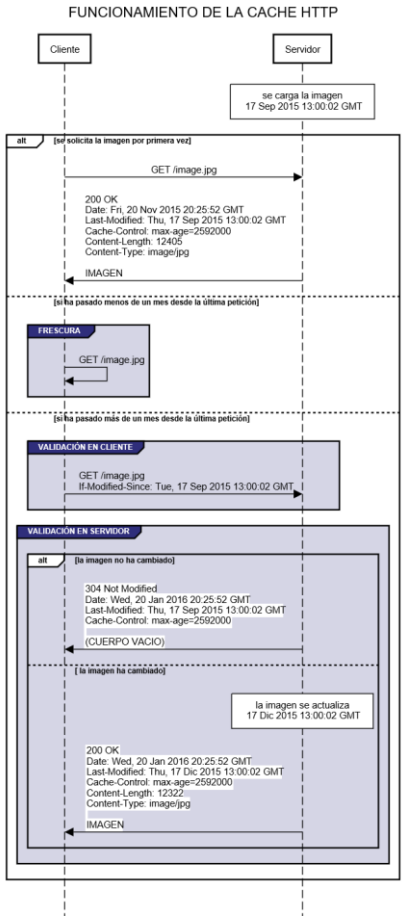
```
datos_codificados = urllib.parse.urlencode(datos)
```

3. (1 punto) En el protocolo OAuth 2.0 para aplicaciones de escritorio, una vez que el usuario se ha identificado correctamente y ha validado los permisos de la aplicación, ¿cómo consigue enviar el servidor de autenticación y autorización una respuesta a la aplicación? ¿Qué se envía en dicha respuesta? Apoya tu respuesta en un diagrama de secuencia que muestre las interacciones HTTP que tienen lugar entre los actores implicados en este paso.

Mediante una respuesta con código 302 cuya cabecera *Location* apunta a la *redirect_uri* de la aplicación. La *redirect_uri* contiene el *auth_code* necesario para obtener el *access_token* en una petición posterior.



4. (1 punto) Describe los mecanismos de gestión de la cache del protocolo HTTP mediante un ejemplo representado en un diagrama de secuencia. Indica cuáles son las cabeceras más significativas en cada interacción entre cliente y servidor.

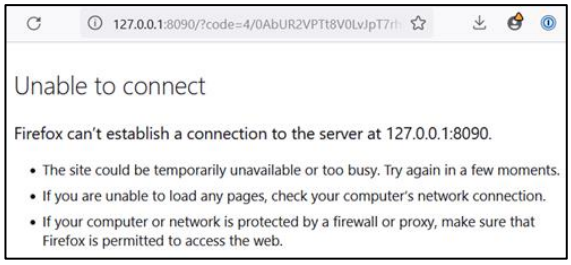


5. (0,5 puntos) ¿Qué estrategias utilizarías en una aplicación web en Tomcat para compartir información entre diferentes usuarios conectados?
BD o contexto dependiendo de la necesidad de remanencia de la información.

6. (0,5 puntos) El siguiente segmento de código gestiona la respuesta del servidor de autenticación y autorización mediante OAuth 2.0 en una aplicación de escritorio.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 8090))
server_socket.listen(1)
print("\t\tSocket listening on port 8090")

print("\t\tWaiting for client requests...")
client_connection, client_address = server_socket.accept()

peticion = client_connection.recv(1024).decode()
print("\t\tSe ha recibido del navegador la siguiente solicitud:")
print("\n" + peticion)

primera_linea = peticion.split('\n')[0]
aux_auth_code = primera_linea.split(' ')[1]
auth_code = aux_auth_code[7:].split('&')[0]
print("auth_code: " + auth_code)

client_connection.close()
server_socket.close()
```

La aplicación ejecuta correctamente el segmento de código mencionado (es decir, se obtiene el *auth_code* y la ejecución de la aplicación sigue adelante sin que se produzca ningún error), pero el navegador muestra la siguiente salida:



¿A qué se debe? ¿Cómo lo solucionarías?

En el servidor implementado mediante un socket, éste se cierra después de obtener el *auth_code* sin devolver una respuesta HTTP al navegador. Una posible solución:

```
print("auth_code: " + auth_code)

# erabiltzaileari erantzun bat bueltatu
http_response = """\
HTTP/1.1 200 OK

<html><head><title>Proba</title></head><body>The authentication flow has
completed. Close this window.</body></html>
"""
client_connection.sendall(str.encode(http_response))

client_connection.close()
```

7. (1 punto) En el siguiente segmento de código HTML se declaran las entradas correspondientes a una tarea y un fichero de una práctica en eGela.

```
<li class="activity assign modtype_assign " id="module-6256046">
  <div>
    <div class="mod-indent-outer w-100">
      <div class="mod-indent mod-indent-1">
      </div>
      <div>
        <div class="activityinstance">
          <a class="aalink dimmed conditionalhidden" onclick="" href="https://egela.ehu.eus/mod/assign
          /view.php?id=6256046">
          <img src="https://egela.ehu.eus/theme/image.php/ehu/assign/1684226285/icon" class="iconlarge
          activityicon" alt="" role="presentation" aria-hidden="true">
            <span class="instancename">
            GRUPO 1 - Entrega Practica 1- Cliente IoT
              <span class="accesshide ">
              Tarea
              </span>
            </span>
          </a>
        </div>
        <div class="availabilityinfo ishidden">
          <span class="badge badge-info">
          No mostrado a los estudiantes
          </span>
        </div>
      </div>
    </div>
  </div>
</li>
<li class="activity resource modtype_resource " id="module-6256043">
  <div>
    <div class="mod-indent-outer w-100">
      <div class="mod-indent mod-indent-1">
      </div>
      <div>
        <div class="activityinstance">
          <a class="aalink" onclick="" href="https://egela.ehu.eus/mod/resource/view.php?id=6256043">
          <img src="https://egela.ehu.eus/theme/image.php/ehu/core/1684226285/f/pdf" class="iconlarge
          activityicon" alt="" role="presentation" aria-hidden="true">
            <span class="instancename">
            Cliente IoT Guión
              <span class="accesshide ">
              Archivo
              </span>
            </span>
          </a>
        </div>
      </div>
    </div>
  </div>
</li>
```

Para extraer la URI de la tarea se utiliza el siguiente segmento de código Python. Rellena los huecos sin utilizar imágenes como elementos de apoyo en la búsqueda.

Una posible solución:

```
print("\n### Buscando tareas de prácticas de laboratorio... ###")
html = BeautifulSoup(respuesta.content, 'html.parser')
resultados = html.find_all('span', { 'class': 'instancename'})
for each in resultados:
    elemento = each.find('span', { 'class': 'accesshide' })
    if elemento.get_text() == 'Tarea':
        print("\n### Se ha encontrado una tarea! ###")
        uri = each.parent['href']
```

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

```
uri = 'https://www.googleapis.com/calendar/v3/calendars/' + calendarID + '/events'
headers = {'Host': 'www.googleapis.com',
           'Content-Type': 'application/json'}
data = {'end': {'date':'2017-05-27'},
        'start': {'date':'2017-05-26'},
        'description': 'Entregar Trabajo 3',
        'summary': 'SW'}
response = requests.post(uri, headers=headers, data=data)
```

Identifica los 3 errores en el código suponiendo que todas las variables utilizadas han sido previamente declaradas antes de usarse. NOTA: los tres errores no son sintácticos, sino conceptuales.

1.- Codificar los datos en formato JSON.
2.- Falta la cabecera *Content_Length*.
3.- Falta la cabecera *Authentication* con *access_token* de OAuth.

9. (1 punto) Dado el siguiente JSP, escribir el código HTML de la página renderizada en el navegador.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.*" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <%  int i=0;
      for(i=1; i<3; i++) { %>
      <ul id="lista<%= i %>">
          <li>Server: <%= new Date().toString() %></li>
      </ul>
  <% } %>
  <script>
    for(i=1; i<<%= i %>; i++) {
      var data = new Date();
      var txt_elem = document.createTextNode("Client: " + data);
      var li_elem = document.createElement("li");
      li_elem.appendChild(txt_elem);
      ul_elem = document.getElementById("zerrenda<%= i-1 %>")
      ul_elem.appendChild(li_elem);
    }
  </script>
</body>
</html>
```

```
<html>
  <head>
    <title>
    Title
    </title>
  </head>
  <body>
    <ul id="zerrenda1">
      <li>
      Server Date: Wed May 24 07:57:04 CEST 2023
      </li>
    </ul>
    <ul id="zerrenda2">
      <li>
      Server Date: Wed May 24 07:57:04 CEST 2023
      </li>
      <li>
      Client date: Wed May 24 2023 07:57:04 GMT+0200 (Central European Summer Time)
      </li>
      <li>
      Client date: Wed May 24 2023 07:57:04 GMT+0200 (Central European Summer Time)
      </li>
    </ul>
    <script>
    for(i=1;
    i<3;
    i++) {
    var data = new Date();
    var txt_elem = document.createTextNode("Client date: " + data);
    var li_elem = document.createElement("li");
    li_elem.appendChild(txt_elem);
    ul_elem = document.getElementById("zerrenda2") ul_elem.appendChild(li_elem);
    }
    </script>
  </body>
</html>
```

10. (0,5 puntos) Los parámetros *client_id* y *client_secret*

a. Permiten saber si una aplicación está registrada para usar una API web.
b. Constituyen el nombre de usuario y la contraseña utilizados para identificar al programador de una aplicación que hace uso de una API web.
c. Constituyen el nombre de usuario y la contraseña con las que una aplicación accede a los datos de un usuario en un servicio web.
d. Se utilizan para obtener el *access_token* mediante una función hash.

11. (0,5 puntos) Dado el siguiente fichero de despliegue de una aplicación web:

```
<servlet>
        <servlet-name>DoLoginServlet</servlet-name>
        <servlet-class>DoLogin</servlet-class>
</servlet>
<servlet-mapping>
         <servlet-name>DoLogin</servlet>
         <url-pattern>/do/Login</url-pattern>
<servlet-mapping>
```

¿Qué URI será una petición correcta al servlet de nombre DoLoginServlet? Justifica la respuesta.

Ninguna, porque la URI no se encuentra correctamente mapeada a la clase del servlet.

12. (0,5 puntos) Dadas estas dos formas de redireccionamiento:

A. RequestDispatcher rd = request.getRequestDispatcher("/html/pagina.html");
rd.forward(request, response);
B. response.sendRedirect("webServer/html/pagina.html");

Respecto a los mensajes que se intercambian cliente y servidor, ¿cuáles de las siguientes afirmaciones son válidas?

a) Las dos formas envían la página HTML en el cuerpo del mensaje de la respuesta.
b) En B se envía un mensaje 200 con la página HTML en el cuerpo del mensaje
c) En B se envía un mensaje 302 con la URI de la página HTML en la cabecera *Location*.
d) Las dos formas envían la página HTML en el cuerpo del mensaje de la respuesta, pero B la envía truncada.

13. (1 punto) Con la ayuda de la documentación del API de Gmail, escribe la petición HTTP que permite a un usuario autenticado recuperar los mensajes no leídos con remitente sw2023@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).

GET /gmail/v1/users/me/messages?q=sw2023%40gmail.com+is%3Aunread HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer [YOUR_ACCESS_TOKEN]