Sistemas Distribuidos e Internet

1. ## ¿Cómo gestiona Node.js las operaciones de IO?

   De forma asíncrona y sin bloqueo, utiliza un único hilo de ejecución (single threading) que gestiona las entradas y salidas asíncronas de los clientes conectados. Esto pemite ejecutar varios procesos de E/S sin producir un bloqueo en el sistema.

2. ## ¿Cómo se escala una aplicación?

   - Verticalmente, agregar más recursos a una máquina.
   - Horizontalmente, agregar más máquinas para desempeñar la misma tarea. (Es la utilizada por Node.js)

3. ## ¿Qué diferencias hay entre req.body.n y req.query.n?

   Body coge los datos en el cuerpo de la petición (peticiones Post) y query los coge de la URL (peticiones Get).

4. ## ¿Cómo se inserta un atributo en Swig?

   ```
   <td>{{ atributo }}</td>
   ```

5. ## Indica las distintas formas de crear un módulo y exportarlo. (module.exports =)

   - Como una función, ejecutan la función al ser incluidos.
   - Como un objeto, permiten acceder a sus variables y funciones. (Es la que usamos en la práctica por ahora)
   - Como una clase, permiten crear instancias.

6. ## ¿Cómo se accede al atributo id en MongoDB?

   En MongoDB los ids son objetos, no cadenas de caracteres

7. ## ¿Cuál es el objetivo principal de los servicios web?

   La interoperabilidad entre operaciones.

8. ## ¿Qué es el protocolo SOAP?

   Protocolo de comunicación basado en XML para el intercambio de información entre entornos distribuidos.

9. ## ¿Qué diferencia hay entre jQuery y Node.js si los dos utilizan JavaScript?

   El código de jQuery se ejecuta en el lado del cliente y el código de Node.js se ejecuta en el lado del servidor.

10. ## Encuentra si hay fallos en el siguiente fragmento de código.

```
var express = require('express');
var app = express();

express.get('/saludar', function() {
    res.send('saludar');
});
```

   - El objeto “app” maneja la aplicación, “express” ni siquiera existe.
   - A la función le faltan los parámetros de la petición y la respuesta, normalmente denominados req y res.

11. Completa el siguiente fragmento de código.

```
app.get('/prueba/valor/', function(reg, res){

});
```

Solución:

```
app.get('/prueba/:valor', function (req, res) {
    var respuesta = req.params.valor;
    res.send(repuesta);
});
```

12. El siguiente fragmento de código se corresponde con un servicio Web que retorna una lista de usuarios. ¿Hay algún fallo conceptual?, si es así explicarlos.

```
app.get('/api/usuarios/', function(req, res){
    gestorBD.obtenerUsuarios(, function(usuarios){
        if(usuarios == null){
            res.json({ error : "un error"});
        } else {
            res.send(JSON.stringify(canciones));
        }
    });
});
```

- Tal como está el código, habría que mandar un JSON al método “obtenerUsuarios” aunque sea vacío. Debería ser:

```
gestorBD.obtenerUsuarios({}, function(usuarios){
```

- En ningún sitio está declarada una variable denominada “canciones”.

13. ¿En qué orden imprimirá el siguiente fragmento de código?

```
var usuario = { nombre : "uniovi" }
console.log('A');
mongo.connect(app.get('db'), function(err, db){
    var collection = db.collection('can');
    console.log('B');
    collection.insert(can, function(err, out){
        console.log('C');
        res.send("id: " + out.ops[0]._id);
        db.close();
    });
    console.log('D');
});
console.log('E');
```

AEBDC

14. ¿Cómo se realiza la gestión de dependencias en Node.js?

Con NPM (Node Package Manager).

15. ¿Cuál es el fichero equivalente del pom.xml de SpringBoot en Node.js? ¿Qué información del proyecto contiene?

Es el fichero package.json. En él se definen:

- La configuración y los metadatos de la aplicación.
- Las dependencias utilizadas.

16. ¿Cómo se llaman a las dependencias instaladas, también denominados módulos, en el proyecto para su uso en este?

```
var my_package = require('<package_name');
```

17. ¿Qué características tiene una comunicación síncrona?

- Las operaciones son bloqueantes.
- Las operaciones se ejecutan de forma secuencial.
- El programa permanece bloqueado hasta que la operación termine.
- El cliente espera la respuesta del servidor para continuar con el flujo del programa.

18. ¿Qué características tiene una comunicación asíncrona?

- Las operaciones no son bloqueantes, no se espera a la terminación de una operación para seguir el flujo del programa.
- Se realizan mediante el sistema de callbacks (retrollamadas) y/o promesas.
- Si un proceso de I/O tarda demasiado, permite que continúe otro proceso antes de que la transmisión haya finalizado.

19. ¿Cuál es el framework de desarrollo de aplicaciones web que usamos con Node.js?

Express

20. ¿Qué es el enrutamiento o Routing en Express?

- Define los puntos finales de una aplicación (URI), como se van a procesas las peticiones (request) y las respuestas que se le enviaran al cliente (response).
- Para ello se implementa el código de la siguiente manera, app.METHOD(PATH, HANDLER)
    - app: instancia de una aplicación express.
    - METHOD: método de solicitud HTTP (get o post).
    - PATH (ruta): vía de acceso al servidor (URL introducida). Admiten el uso de comodines y otras expresiones regulares.
    - HANDLER (manejador): función que se ejecuta cuando se recibe la petición. Tiene acceso al objeto petición (req) y al objeto respuesta (res).

21. ¿Qué métodos de respuestas usamos en Express?

- send(): envía una respuesta en forma de cadena.
- json(): envía una respuesta en formato JSON.
- redirect(): redirecciona otra URL.

22. ¿Cómo se organiza los módulos de una aplicación Express?

- Routes: almacena los módulos de rutas (Controladores).
- Public: almacena los ficheros estáticos. (Imágenes, videos, scripts, css, etc.).
- Views: almacena las plantillas que se utilizan en la aplicación. (HTML)
- Modules: almacena los módulos de base de datos (Repositorios)

23. ¿Cómo se utilizan las clases que se usan en el módulo "routes"?

- Se utiliza la sentencia "module.exports".
    - Pueden recibir parámetros en su constructor.

```
module.exports = function (app, swig, usersRepository,
requestsRepository) { … };
```

- Para incluirlos en la aplicación usamos "require(fichero)(parámetros)".

```
require("./routes/rUsers.js")(app, swig, usersRepository,
requestsRepository);
```

24. ¿De qué tipo son los valores que se obtiene a través "req.query"?

Son siempre cadenas de texto.

25. ¿Qué diferencia hay entre "req.query" y "req.param"?

- req.query: recoge los parámetros cuando se encuentran en la URL usando una combinación de clave-valor.
- req.param: recoge los parámetros cuando se encuentran embebidos, es decir sin especificar la clave.

26. ¿Cómo se indican las estructuras de control en Swig?

Con {% for <variable temporal> in <colección>%} e indicando el final de estas, {% endfor %}

27. ¿Cómo se indican las estructuras condicionales en Swig?

Con {% if <condición> %} e indicando el final de estas, {% endif %}

28. ¿Qué es un bloque en Swig, para que se usa y cómo se utiliza?

Swig ofrece un sistema de composición de plantillas basado en herencia y redefinición de bloques, es decir, permite definir bloques en una plantilla padre que podrán ser redefinidos por sus hijos. (Como el patrón Decorator)

Para ellos se definen estos bloques en el padre:

{% block <nombreDelBloque>%} Contenido {% endblock %}

Para crear los hijos se indica de la siguiente forma:

{% extends "padre.html" %}

29. ¿Cómo se les pasan datos a las plantillas de Swig?

En los archivos del módulo routes debemos devolver en la respuesta la función de Swig, renderFile(<plantilla>, <datosUsadosEnLaPlantilla>)

30. ¿Qué es un BSON?

Es una versión ligera de JSON usada por MongoDB.

31. ¿Qué son los documentos y colecciones de MongoDB?

- Documentos: es un registro de MongoDB, como una fila en una tabla de una base de datos relacional.
- Colecciones: son carpetas donde se almacenan los documentos, algo parecido a una tabla en base de datos relacional.

32. ¿Qué es un ObjectID?

Es un id que crea automáticamente MongoDB, funciona como identificador único del documento. NO ES UN STRING, ES UN OBJETO.

33. ¿Qué es el array "ops" en una función de MongoDB?

Es un array con los documentos insertados.

34. ¿Cómo se accede a la sesión en Express?

A través de cualquier petición con el atributo "session" (req.session).

35. ¿Qué es un enrutador?

Permite definir funciones que procesan las peticiones.

- Ejecutar cualquier lógica de negocio.
- Dejar correr la petición para que la procese el siguiente elemento.
- Cortar la petición.

Se debe agregar el enrutador antes de la definición de los controladores que controlan.

36. ¿Qué es la arquitectura REST?

- REST (Representational State Transfer) es un estilo para sistemas hipermedias distribuidos (WWW).
- Los datos y la funcionalidad se consideran recursos y se accede a ellos utilizando identificadores de recursos uniformes (URI).
- Suelen gestionar datos CRUD (Create Read Update Delete).
- Cada recurso debería tener una URl.

37. ¿Qué es un sistema RESTful?

Es un sistema que sigue los principios REST.

38. ¿Cuáles son los métodos HTTP para servicios RESTful?

- POST: crear datos.
- GET: leer datos.
- PUT: actualizar reemplazando datos.
- PATCH actualizar cambiando los datos.
- DELETE: borrar datos.

39. ¿Cuáles son las características de REST?

- Protocolo cliente/servidor sin estado, es decir, sin sesión, cookies ni reescritura de URL.
- Interfaz uniforme.
    - Conjunto de operaciones bien definidas.
    - Sintaxis universal para identificar los recursos (URI).
    - Mensajes intercambiados son autodescriptivos.
    - La manipulación del recurso se realiza a través de su representación.
- Sistema de capas. Cada una de las capas lleva a cabo una funcionalidad.
- Uso de hipermedios. Cumple el principio HATEOAS.
- Infraestructura de almacenamiento en cache. Los recursos deben ser declarados como cacheables o no cacheables.

40. ¿Qué es el principio HATEOAS?

Un recurso REST puede contener enlaces a otro recurso REST.

41. ¿Cómo devolvemos el código de repuesta en una API REST?

En el atributo status (res.status).

42. ¿Cómo transformamos objetos a JSON en Node.js?

- JSON.stringfy(objeto)
- res.json

43. ¿Cómo se realiza la identificación de usuarios en RESTful?

Utilizando un token de seguridad que acompaña a las peticiones y permite controlar los permisos de los usuarios.

44. ¿Qué mecanismos basados en tokens existen?

- Token único:
    - Cada usuario es provisto de un token único que le identifica y es asociado a todas las peticiones realizadas a los servicios.
- Token por login:
    - Cuando el cliente envía sus credenciales a un servicio específico recibe un token.
    - Este debe ser almacenado y enviado en todas las peticiones que el cliente realiza.
    - Puede caducar:
        - Después de un tiempo sin usarlo.
        - En cada petición y reenviando uno nuevo.

45. ¿Qué es un cliente REST?

Es una aplicación que consuma un servicio web REST.

46. ¿Qué es CORS?

Intercambio de recursos de origen cruzado, es un mecanismo que utiliza para añadir cabeceras adicionales HTTP para permitir que un programa obtenga permiso para acceder a recursos seleccionados desde un servidor de origen distinto (dominio) al que pertenece.

47. ¿Para qué se usa el control de acceso HTTP Access-Control-Allow-Origin?

Indica si la respuesta puede ser compartida con recursos del ORIGIN dado.

48. ¿Qué es el ORIGIN?

Es el origen del contenido web que se define por el esquema (protocolo), el host (dominio) y el puerto de la URL utilizada para acceder a él.

49. ¿Para qué se usa el control de acceso HTTP Access-Control-Allow-Credentials?

Indica si el servidor permite que las credenciales del usuario se incluyan en solicitudes de origen cruzado. Estos credenciales son cookies, encabezados de autorización o certificados de cliente TLS.

50. ¿Para qué se usa el control de acceso HTTP Access-Control-Allow-Methods?

Especifica los métodos permitidos al acceder al recurso en respuesta a una solicitud.

51. ¿Para qué se usa el control de acceso HTTP Access-Control-Allow-Headers?

Indica que encabezados HTTP se pueden usar durante la solicitud actual.

52. ¿Qué es Envelope?

Es el contenedor del mensaje SOAP. Contiene:

- Header, cabecera que es opcional.
- Body, cuerpo que es obligatorio.

53. ¿Qué es SOAP Header?

Es un elemento opcional para incluir meta-información sobre el mensaje SOAP, como debe ser procesado el mensaje.

54. ¿Qué es SOAP Body?

Contiene la información relativa a la llamada / respuesta.

- Datos enviados como repuesta tras la ejecución de servicio.
- Información de error y estado.

55. ¿Cuáles son las desventajas de servicio Web SOAP?

- Al usar gramática XML puede ser más lento que otros.
- Los binarios se codifican en texto, si el archivo es demasiado grande ralentizará el rendimiento.

56. ¿Qué es WSDL?

Es un lenguaje basado en XML utilizado para describir la funcionalidad que proporciona un servicio Web, es decir, define como serán los mensajes SOAP que se utilizarán.

57. ¿Cuál es la estructura de un documento WSDL?

Consta de dos partes:

- Definición de la interfaz del servicio:

  - Tipos de datos - <types>: define los tipos de datos utilizados
  - Mensajes - <message>: descripción de los mensajes de entrada/salida que serán utilizados en las operaciones.
  - Tipo de puerto - <portType>: especificación de las operaciones (funcionalidad) soportadas por un Endpoint de un servicio.
- Definición de la implementación del servicio:
  - Uniones - <binding>: especifica cómo se enlaza la operación con un protocolo.
  - Servicio - <service>: indica donde se encuentra el servicio.

58. ¿Qué es un Endpoint?

Define los puntos de entradas a los que los clientes podrán hacer peticiones al servicio web disponible.

59. ¿Qué diferencia hay en SOAP y REST?

- SOAP consume más recursos que REST al utilizar mensajes más pesados.
- SOAP tiene gran cantidad de aplicación de soporte y generación automática de clientes y pruebas.
- SOAP, cono los descriptores WSDL el cliente puede conocer en detalle las operaciones, en cambia en REST eso depende de la calidad de la documentación.