Demo

Submit this form with test data, and Form Publisher will generate a test invoice for you that you will receive via email. You can then edit all the settings you want to customize your workflow.

¿Cuál de las siguientes afirmaciones sobre Node.js es correcta? 0 puntos

- [x] Node.js utiliza un modelo de operaciones de E/S sin bloqueo (non-blocking) para reducir la latencia.
- [ ] Node.js ejecuta JavaScript en el navegador.
- [ ] Node.js no permite la ejecución de operaciones de E/S de manera simultánea.

¿Cuál de las siguientes afirmaciones es cierta sobre la arquitectura modular en Node.js? 1 punto

- (●) Los módulos definidos como funciones se ejecutan automáticamente cuando se incluyen en la aplicación.
- ( ) Los módulos definidos como clases permiten crear múltiples instancias, pero no pueden tener métodos.
- ( ) Los módulos definidos como objetos se comportan como clases y permiten instanciar nuevos objetos.
- ( ) Los módulos definidos como funciones requieren un método new para inicializarlos.

¿Qué característica hace que MongoDB sea diferente de las bases de datos relacionales tradicionales? 1 punto

- [x] No utiliza esquemas predefinidos, permitiendo estructuras de datos dinámicas y flexibles.
- [ ] Almacena datos en tablas y columnas con relaciones fijas.
- [ ] Está diseñada para transacciones ACID desde el principio.
- [ ] Utiliza un modelo de operaciones sincrónicas para acceder a los datos.

¿Cuál de las siguientes opciones describe una función middleware en Express? 1 punto

- [x] Una función que se ejecuta antes de llegar al controlador y puede modificar la solicitud o la respuesta, o bien cortar la solicitud.
- [ ] Una función que gestiona operaciones de E/S síncronas antes de llegar al controlador.
- [ ] Una función que ejecuta el Event Loop y permite la ejecución de código de forma secuencial.
- [ ] Una función que permite crear módulos para un uso específico dentro de una aplicación.

¿Qué hace el módulo crypto en Node.js? 1 punto

- [x] Permite cifrar y descifrar datos, lo cual es útil para la seguridad y el manejo de contraseñas.
- [ ] Controla el acceso a módulos de la aplicación
- [ ] Gestiona operaciones asíncronas relacionadas con bases de datos.
- [ ] Se usa para autorizar y autenticar usuarios en una aplicación Express.

¿Cuál de las siguientes afirmaciones sobre el uso de certificados en un servidor HTTPS es correcta? 1 punto

- [x] Los certificados se utilizan para cifrar las comunicaciones entre el servidor y el navegador, proporcionando seguridad adicional.
- [ ] Los certificados son opcionales, pero se recomiendan para mejorar el rendimiento del servidor.
- [ ] Los certificados solo se requieren para servidores de bases de datos MongoDB.
- [ ] Los certificados son necesarios solo si se usa un sistema operativo Linux.

¿Cuál es la diferencia entre la escalabilidad horizontal y la vertical en Node.js? 0 puntos

- [ ] La escalabilidad horizontal agrega más recursos a un solo nodo, mientras que la escalabilidad vertical agrega más nodos a un sistema.
- [ ] La escalabilidad horizontal permite ejecutar múltiples hilos, mientras que la escalabilidad vertical usa un solo hilo para todas las peticiones.
- [x] La escalabilidad horizontal agrega más nodos a un sistema, mientras que la escalabilidad vertical agrega más recursos a un solo nodo.
- [ ] La escalabilidad horizontal es el método preferido por Node.js, mientras que la escalabilidad vertical no es posible.

¿Qué módulo necesitas para obtener parámetros de un formulario enviado con el método HTTP POST en Node.js? 0 puntos

- [ ] fs
- [ ] path
- [x] body-parser
- [ ] os

¿Cuál es el beneficio principal del modelo de operaciones de E/S asíncronas en Node.js? 0 puntos

- [x] Las operaciones se pueden ejecutar sin esperar a que terminen, permitiendo un mayor rendimiento y capacidad de respuesta.
- [ ] Las operaciones se ejecutan en orden secuencial, lo que garantiza que el código se ejecute de manera consistente.
- [ ] El modelo de E/S asíncrona permite usar múltiples hilos, mejorando la capacidad del sistema.
- [ ] El modelo de E/S asíncrona es más adecuado para operaciones que requieren mucho tiempo de CPU.

¿Cuál de las siguientes declaraciones sobre el Event Loop de Node.js es verdadera? 0 puntos

- [ ] El Event Loop en Node.js permite la ejecución de múltiples hilos para gestionar peticiones concurrentes.
- [x] El Event Loop es responsable de gestionar las operaciones de E/S asíncronas y permite mantener la aplicación reactiva.
- [ ] El Event Loop es exclusivo para operaciones sincrónicas y bloqueantes.
- [ ] El Event Loop solo funciona cuando Node.js se ejecuta en sistemas operativos Linux.

¿Qué comando se usa para instalar un paquete en Node.js y agregarlo como dependencia al archivo package.json? 0 puntos

- [ ] npm install <nombre_del_paquete> --no-save
- [x] npm install <nombre_del_paquete> --save
- [ ] npm install <nombre_del_paquete> -g
- [ ] npm install <nombre_del_paquete> --no-save --global

¿Qué es el archivo package.json en un proyecto Node.js? 0 puntos

- [x] Es un archivo de configuración que define las dependencias, scripts y metadatos del proyecto.
- [ ] Es el archivo principal donde se almacena la lógica de la aplicación.
- [ ] Es el archivo que contiene la configuración del servidor HTTP
- [ ] Es el archivo que controla el flujo del Event Loop en Node.js

¿Qué se entiende por "comunicación asíncrona" en Node.js? 0 puntos

- [ ] Las operaciones se ejecutan en paralelo, permitiendo múltiples flujos de ejecución al mismo tiempo.
- [ ] Las operaciones utilizan múltiples hilos para realizar tareas simultáneas.
- [x] Las operaciones no son bloqueantes, permitiendo que otras tareas continúen ejecutándose mientras se esperan resultados.
- [ ] Las operaciones se ejecutan secuencialmente y bloquean el flujo de ejecución hasta que se completan.

¿Cuál es la ventaja de utilizar express.static() en una aplicación Node.js? 0 puntos

- [x] Permite servir archivos estáticos (como imágenes, CSS, o JavaScript) directamente desde un directorio específico.
- [ ] Permite crear servidores HTTP.
- [ ] Facilita el manejo de operaciones de E/S asíncronas.
- [ ] Crea rutas dinámicas en la aplicación.

¿Qué comando se usa para instalar una versión específica de un paquete en Node.js? 0 puntos

- ○ npm install <nombre_del_paquete> --save
- ○ npm install <nombre_del_paquete> --global
- ◉ npm install <nombre_del_paquete>@<versión>
- ○ npm install <nombre_del_paquete> --no-save

¿Cuál es la diferencia entre npm install --save y npm install --global? 0 puntos

- ◉ --save agrega el paquete como dependencia en el package.json del proyecto, mientras que --global instala el paquete globalmente para usarlo en cualquier proyecto.
- ○ --save instala el paquete para uso local, mientras que --global instala el paquete para uso global.
- ○ --save instala el paquete temporalmente, mientras que --global lo instala permanentemente.
- ○ --save instala el paquete sin agregarlo a package.json, mientras que --global lo instala y lo agrega a package.json.

¿Qué resultado esperas cuando ejecutas npm start en un proyecto Node.js? 0 puntos

- ◉ Inicia el servidor definido en el script start del archivo package.json
- ○ Inicia un servidor web predeterminado de Node.js.
- ○ Compila el código del proyecto y muestra errores si los hay
- ○ Instala todas las dependencias necesarias para el proyecto.

¿Qué hace la función res.json() en Node.js? 0 puntos

- ○ Establece el código de estado de la respuesta.
- ◉ Envía una respuesta en formato JSON al cliente
- ○ Redirecciona la solicitud a otra URL.
- ○ Envía un archivo como respuesta.

¿Qué motor de JavaScript utiliza Node.js para ejecutar su código? 0 puntos

- ○ JavaScriptCore
- ○ SpiderMonkey
- ○ Chakra
- ◉ V8

¿Qué permite el uso de módulos en Node.js? 0 puntos

- ◉ Dividir la funcionalidad de la aplicación en componentes reutilizables y facilitar la gestión de dependencias.
- ○ Compilar el código JavaScript a un formato ejecutable.
- ○ Ejecutar múltiples hilos para operaciones concurrentes.
- ○ Crear servidores web con alta escalabilidad.

¿Cuál es el uso de express.Router() en una aplicación Node.js? 0 puntos

- ○ Permite definir una nueva instancia del servidor HTTP.
- ○ Se usa para crear una interfaz gráfica para el usuario.
- ○ Se utiliza para administrar operaciones de E/S sin bloqueo.
- ◉ Permite la definición de rutas en un objeto independiente, facilitando la modularidad y el diseño limpio de las rutas

¿Cuál es la principal razón por la que Node.js no es adecuado para aplicaciones intensivas en CPU? 0 puntos

- ○ Node.js tiene un modelo de operaciones de E/S síncronas, que no es eficiente para cálculos pesados.
- ◉ Node.js utiliza un solo hilo para ejecutar el código, lo que puede causar bloqueos si se realizan operaciones intensivas en CPU.
- ○ Node.js no admite operaciones de E/S asíncronas.
- ○ Node.js está diseñado solo para aplicaciones de tiempo real.

¿Cuál es la diferencia entre npm install <nombre_del_paquete> --save-dev y npm install <nombre_del_paquete> --save? 0 puntos

- ○ --save-dev instala el paquete globalmente, mientras que --save lo instala localmente
- ○ --save-dev instala el paquete temporalmente, mientras que --save lo instala permanentemente
- ◉ --save-dev agrega el paquete como dependencia para el desarrollo, mientras que --save lo agrega como dependencia de producción.
- ○ --save-dev agrega el paquete como dependencia para producción, mientras que --save lo agrega como dependencia de desarrollo.

¿Cuál es el propósito de process.env en Node.js? 1 punto

- ○ Opción 1
- ◉ Acceder a las variables de entorno del proceso Node.js, lo que permite configurar el entorno de ejecución.
- ○ Determinar el entorno operativo de Node.js, como el sistema operativo y la arquitectura.
- ○ Crear nuevas variables de entorno para compartir entre diferentes procesos.
- ○ Configurar el Event Loop para operaciones sincrónicas.

¿Cuál es el uso de process.exit() en Node.js? 1 punto

- ◉ Terminar el proceso actual, devolviendo un código de salida opcional.
- ○ Interrumpir una operación de E/S en curso.
- ○ Reiniciar el proceso Node.js.
- ○ Detener el Event Loop y finalizar todas las operaciones.

¿Qué es un Promise en Node.js? 1 punto

- ◉ Un objeto que representa la eventual resolución o rechazo de una operación asíncrona, permitiendo un manejo más estructurado de las operaciones.
- ○ Un tipo especial de función que siempre se ejecuta de manera sincrónica.
- ○ Un objeto que asegura que una operación sincrónica se complete antes de continuar con otras tareas.
- ○ Una estructura utilizada para el manejo de errores en operaciones de E/S.

¿Cuál es la diferencia entre setImmediate() y setTimeout() en Node.js? 1 punto

- [x] setImmediate() ejecuta una función al comienzo del siguiente ciclo del Event Loop, mientras que setTimeout() la ejecuta después de un tiempo determinado.
- [ ] setImmediate() es para operaciones sincrónicas, mientras que
- [ ] setImmediate() crea un nuevo hilo, mientras que setTimeout() ejecuta en el hilo principal.
- [ ] setImmediate() bloquea el Event Loop, mientras que setTimeout() permite la ejecución de otras tareas mientras se espera

cual es la raz 1 punto

- [x] exports es un alias para module.exports, pero para exportar un objeto completo, se debe usar module.exports.
- [ ] exports y module.exports son completamente diferentes; uno es para exportar funciones y el otro para objetos.
- [ ] exports se usa para exportar objetos, mientras que module.exports se usa para exportar funciones.
- [ ] exports se usa para exportar módulos globales, mientras que module.exports se usa para módulos locales.

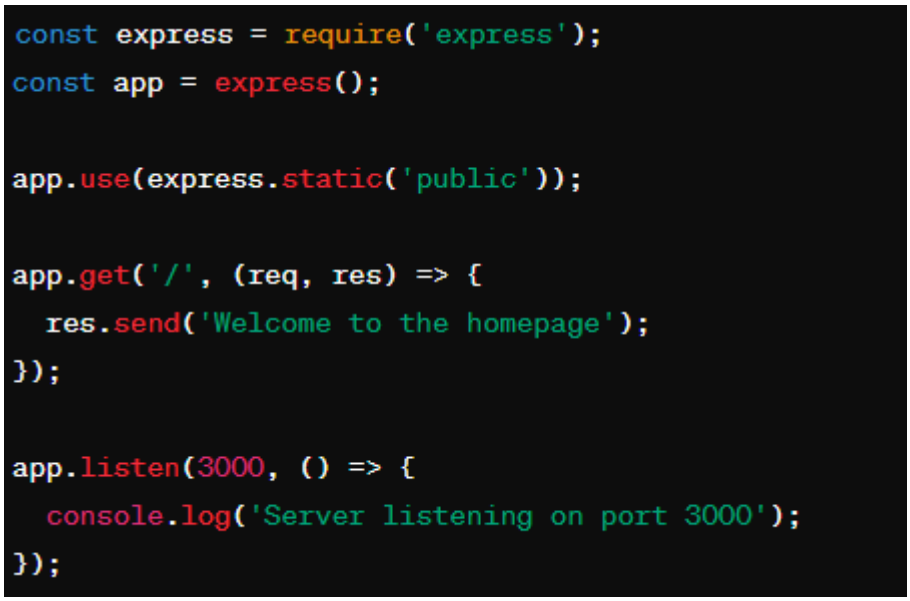
¿Qué es un middleware en Express? 1 punto

- [x] Una función que se ejecuta durante el ciclo de vida de una solicitud y puede modificar la solicitud o la respuesta antes de llegar al controlador.
- [ ] Un módulo adicional que se instala para mejorar el rendimiento del servidor.
- [ ] Una capa intermedia que conecta Node.js con otras aplicaciones.
- [ ] Una función que se ejecuta solo para solicitudes POST.

¿Cuál es el uso de async/await en Node.js? 1 punto

- (●) Permite escribir código asíncrono de una manera más clara y legible, simulando un flujo sincrónico.
- ( ) Se usa para crear múltiples hilos para operaciones concurrentes.
- ( ) Se usa para manejar errores en operaciones asíncronas.
- ( ) Se usa para bloquear el Event Loop hasta que todas las operaciones se completen.

¿Qué hará el siguiente código en un servidor Express? 1 punto



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

app.use(express.static('public'));

app.get('/', (req, res) => {
  res.send('Welcome to the homepage');
});

app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

- (●) Servirá archivos estáticos desde el directorio "public" y responderá con un mensaje de bienvenida en la raíz ('/').
- ( ) Creará un servidor que solo responde con un mensaje de bienvenida.
- ( ) Servirá archivos estáticos desde el directorio "public" y redireccionará la solicitud a la raíz.
- ( ) Responderá solo con archivos estáticos del directorio "public".

¿Qué comando se usa para ejecutar un script definido en el archivo package.json? 1 punto

- (●) npm run <nombre_del_script>
- ( ) npm install <nombre_del_script>
- ( ) npm start
- ( ) npm build <nombre_del_script>

¿Qué es un "callback" en Node.js? 1 punto

- (●) Una función que se pasa como argumento a otra función y se ejecuta después de que una operación se complete.
- ( ) Una función que se ejecuta inmediatamente después de la declaración de una operación sincrónica.
- ( ) Una función que se usa para manejar errores en operaciones asíncronas.
- ( ) Una función que se usa para configurar el Event Loop.

¿Qué hace el siguiente fragmento de código en Node.js? 1 punto

```
const fs = require('fs');

fs.writeFileSync('example.txt', 'Hello, World!');
console.log('File has been written');
```

- (●) Escribe el texto "Hello, World!" en el archivo "example.txt" de forma sincrónica y luego muestra un mensaje en la consola.
- ( ) Crea un archivo llamado "example.txt" y escribe "Hello, World!" en él de forma asíncrona.
- ( ) Lanza un error porque el método writeFileSync no está permitido en Node.js.
- ( ) Lanza un error porque fs no admite operaciones de escritura.

¿Qué salida esperas del siguiente código de Node.js? 1 punto

```
console.log('Start');

setTimeout(() => {
  console.log('After 1 second');
}, 1000);

console.log('End');
```

- (●) 'Start', 'End', 'After 1 second'
- ( ) 'Start', 'After 1 second', 'End'
- ( ) 'End', 'Start', 'After 1 second'
- ( ) 'After 1 second', 'Start', 'End'

¿Qué diferencia existe entre process.exit() y process.kill() en Node.js? 1 punto

- (●) process.exit() finaliza el proceso actual, mientras que process.kill() finaliza un proceso específico por su ID.
- ( ) process.exit() solo finaliza procesos hijos, mientras que process.kill() finaliza cualquier proceso.
- ( ) process.exit() se usa para finalizar procesos asíncronos, mientras que process.kill() finaliza procesos sincrónicos.
- ( ) process.exit() cierra procesos de servidor, mientras que process.kill() cierra procesos locales.

¿Qué resultado produce el siguiente código de Express? 1 punto

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

app.get('/greet/:name', (req, res) => {
  const name = req.params.name;
  res.send(`Hello, ${name}!`);
});

app.listen(3000);
```

- (●) Devuelve un saludo personalizado usando el parámetro de la ruta, como "Hello, John!" cuando se accede a '/greet/John'.
- ( ) Devuelve siempre "Hello, World!" sin importar el parámetro de la ruta.
- ( ) Devuelve un error porque req.params.name no está definido.
- ( ) Devuelve un mensaje de error cuando el nombre no está en la ruta.

¿Qué hace el siguiente fragmento de código en Node.js? 1 punto

```
const { exec } = require('child_process');

exec('ls', (error, stdout, stderr) => {
  if (error) {
    console.error(`Error: ${error.message}`);
    return;
  }
  if (stderr) {
    console.error(`STDERR: ${stderr}`);
    return;
  }
  console.log(`STDOUT: ${stdout}`);
});
```

- (●) Ejecuta el comando 'ls' en un proceso hijo y muestra el resultado en la consola.
- ( ) Crea un nuevo proceso hijo que no hace nada y devuelve un error.
- ( ) Crea un nuevo proceso hijo que muestra un mensaje de error.
- ( ) Ejecuta el comando 'ls' de forma sincrónica y muestra el resultado en la consola.

¿Qué resultado produce el siguiente código en un servidor Express? 1 punto

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

app.use((req, res, next) => {
  console.log('Middleware');
  next();
});

app.get('/', (req, res) => {
  res.send('Home');
});

app.listen(3000);
```

- (●) Imprime "Middleware" en la consola cada vez que se recibe una solicitud y responde con "Home" para la ruta raíz.
- ( ) Responde con "Middleware" para cualquier solicitud entrante.
- ( ) Devuelve un error porque no hay un controlador para la ruta raíz.
- ( ) Devuelve "Middleware" y luego "Home" para la ruta raíz.

¿Qué hace el siguiente fragmento de código de Node.js? 1 punto

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!');
});

server.listen(8080, () => {
  console.log('Server is running on port 8080');
});
```

- (●) Crea un servidor HTTP que responde con "Hello, World!" y escucha en el puerto 8080.
- ( ) Crea un servidor HTTP que responde con "Hello, World!" solo si la solicitud es POST.
- ( ) Crea un servidor HTTP que responde con un archivo estático.
- ( ) Crea un servidor HTTP que no responde a ninguna solicitud

¿Qué resultado se espera del siguiente fragmento de código en Node.js? 1 punto

```
setTimeout(() => {
  console.log('First');
}, 2000);

setImmediate(() => {
  console.log('Second');
});

console.log('Third');
```

- (●) Third', 'Second', 'First'
- ( ) Second', 'Third', 'First'
- ( ) Third', 'First', 'Second'
- ( ) 'First', 'Third', 'Second'

¿Qué hace el siguiente código con express.Router() en Node.js? 1 punto

```
const express = require('express');
const router = express.Router();

router.get('/hello', (req, res) => {
  res.send('Hello from the router');
});

module.exports = router;
```

- (●) Define una ruta que responde con "Hello from the router" cuando se accede a '/hello' y permite modularidad en las rutas.
- ( ) Devuelve siempre "Hello from the router" para cualquier solicitud.
- ( ) Devuelve un error porque express.Router() no está definido.
- ( ) Responde solo con "Hello, World!".

¿Cuál es la diferencia entre un middleware y un controlador en Express? 1 punto

- (●) Un middleware se ejecuta antes de llegar al controlador y puede modificar la solicitud o la respuesta, mientras que un controlador responde directamente a la solicitud.
- ( ) Un middleware siempre envía la respuesta, mientras que un controlador solo procesa la solicitud.
- ( ) Un middleware se usa para manejar errores, mientras que un controlador gestiona operaciones de E/S.
- ( ) Un middleware solo se usa para operaciones sincrónicas, mientras que un controlador se usa para operaciones asíncronas.

¿Cuál es la función de express-session en una aplicación Node.js? 1 punto

- (●) Gestionar las sesiones de los usuarios, permitiendo guardar información sobre usuarios autenticados.
- ( ) Controlar el acceso a archivos estáticos en el servidor.
- ( ) Gestionar las operaciones asíncronas dentro de una aplicación.
- ( ) Controlar el número de instancias de un módulo.

¿Cuál es el propósito de usar app.use() en una aplicación Express? 1 punto

- (●) Agregar middleware o configurar funcionalidades para la aplicación.
- ( ) Agregar nuevos controladores para rutas específicas.
- ( ) Definir rutas para el enrutamiento.
- ( ) Definir las credenciales de acceso para la aplicación.

¿Qué situación podría causar una condición de carrera en una aplicación Node.js con MongoDB? 1 punto

- [x] Cuando se usan operaciones asíncronas que modifican datos sin un mecanismo de sincronización.
- [ ] Cuando se usan operaciones sincrónicas que bloquean el Event Loop
- [ ] Cuando se insertan documentos sin un _id definido.
- [ ] Cuando se usan módulos que no están incluidos en el core de Node.js.

¿Qué ocurre cuando require se usa para importar un módulo definido como un objeto en Node.js? 1 punto

- [x] Todos los require del mismo módulo retornan la misma instancia de objeto, permitiendo compartir estado entre módulos.
- [ ] Cada require del mismo módulo retorna una nueva instancia de objeto.
- [ ] Se crea una única instancia del objeto y se bloquea para acceso concurrente.
- [ ] Los módulos definidos como objetos permiten crear instancias múltiples y distintas.

¿Cuál de las siguientes afirmaciones describe la diferencia entre updateOne() y updateMany() en MongoDB? 1 punto

- [x] updateOne() actualiza el primer documento que coincide con el criterio, mientras que updateMany() actualiza todos los documentos que coinciden con el criterio.
- [ ] updateOne() solo puede actualizar un documento por vez, mientras que updateMany() solo puede insertar nuevos documentos.
- [ ] updateOne() siempre crea un nuevo documento si no encuentra una coincidencia, mientras que updateMany() solo actualiza documentos existentes.
- [ ] updateOne() y updateMany() tienen el mismo comportamiento, solo que updateMany() requiere más recursos del servidor.

¿Cuál es la implicación de utilizar upsert en una operación updateOne() en MongoDB? 1 punto

- [x] Si el criterio de selección no coincide con ningún documento, se crea uno nuevo en la colección.
- [ ] Actualiza todos los documentos que coincidan con el criterio, pero no crea nuevos documentos.
- [ ] Actualiza solo el primer documento encontrado y elimina el resto.
- [ ] Crea un nuevo documento en caso de error con la operación de actualización.

¿Por qué el uso de app.use() en Express puede afectar el orden de ejecución en una aplicación Node.js? 1 punto

- [x] El orden en el que se agregan los middleware y las rutas determina qué código se ejecuta primero, lo que puede afectar el flujo de ejecución.
- [ ] app.use() se utiliza solo para definir rutas específicas, por lo que su orden no afecta el flujo de ejecución.
- [ ] app.use() solo se puede utilizar para agregar controladores y no afecta el orden de ejecución.
- [ ] El orden de app.use() es irrelevante, ya que siempre se ejecutan en paralelo.

¿Qué consecuencias puede tener el uso incorrecto de certificados SSL/TLS en una aplicación Node.js que usa HTTPS? 1 punto

- [x] El navegador podría mostrar advertencias de seguridad o rechazar conexiones si el certificado no es válido o no está emitido por una autoridad confiable.
- [ ] El servidor podría dejar de responder a las solicitudes si los certificados son incorrectos.
- [ ] El certificado podría ser interceptado y utilizado para descifrar datos cifrados.
- [ ] No hay consecuencias significativas, ya que los certificados no afectan la seguridad del servidor.

¿Qué implica el uso de la función next() dentro de un middleware en Express? 1 punto

- [x] Permite que la solicitud continúe hacia el siguiente middleware o controlador, manteniendo el flujo de ejecución.
- [ ] Detiene la ejecución de la solicitud y devuelve un error.
- [ ] Redirige la solicitud a otra ruta sin pasar por el controlador.
- [ ] Permite ejecutar múltiples instancias de un mismo middleware.

¿Qué efecto tiene el uso de skip() y limit() en una consulta MongoDB dentro de una aplicación Node.js? 1 punto

- [x] skip() permite omitir un número específico de documentos en la consulta, y limit() restringe el número de documentos devueltos.
- [ ] skip() permite ejecutar la consulta en paralelo, mientras que limit() restringe el número de hilos utilizados.
- [ ] skip() omite documentos que no cumplen con un criterio específico, y limit() solo devuelve documentos que cumplen con el criterio.
- [ ] skip() permite ejecutar operaciones asíncronas, mientras que limit() garantiza operaciones sincrónicas.

¿Por qué el módulo crypto es crítico para la seguridad en una aplicación Node.js? 1 punto

- [x] Proporciona métodos para cifrar datos y contraseñas, lo que es esencial para proteger información sensible.
- [ ] Controla el acceso a operaciones de I/O y protege contra ataques de denegación de servicio.
- [ ] Permite la autenticación y autorización de usuarios mediante algoritmos de encriptación.
- [ ] Gestiona la comunicación segura entre diferentes módulos en la aplicación.

¿Qué problema puede surgir al usar operaciones asíncronas dentro de un controlador de Express sin manejo adecuado de errores? 1 punto

- [x] Puede provocar condiciones de carrera o comportamiento impredecible si no se manejan adecuadamente las promesas o callbacks.
- [ ] Siempre se detendrá el Event Loop, causando que la aplicación se bloquee.
- [ ] Generará errores sincrónicos que no se pueden manejar adecuadamente.
- [ ] Los errores se manejarán automáticamente por Express sin necesidad de controladores de errores.

¿Qué puede causar una fuga de memoria en una aplicación Node.js? 1 punto

- [x] Acumulación de referencias a objetos que ya no son necesarios, impidiendo que el recolector de basura los elimine.
- [ ] Uso incorrecto de operaciones de E/S sincrónicas, bloqueando el Event Loop.
- [ ] Uso de async/await sin manejar adecuadamente las promesas.
- [ ] Uso excesivo de módulos externos, ocupando demasiada memoria.

¿Qué podría ser un problema con el uso excesivo de variables de aplicación (app.set() y app.get()) en una arquitectura modular de Node.js? 1 punto

- [x] Puede dificultar el mantenimiento y la depuración, ya que las variables de aplicación pueden usarse para almacenar datos globales y causar efectos secundarios inesperados.
- [ ] Puede mejorar la modularidad y el rendimiento, ya que las variables de aplicación se comparten entre módulos.
- [ ] Siempre resulta en errores de concurrencia porque las variables de aplicación son asíncronas.
- [ ] No hay consecuencias negativas al usar variables de aplicación, ya que se consideran parte de las mejores prácticas.

¿Qué diferencia existe entre un módulo definido como función y un módulo definido como clase en Node.js? 1 punto

- [x] Un módulo definido como función se ejecuta automáticamente cuando se incluye, mientras que un módulo definido como clase permite crear instancias.
- [ ] Un módulo definido como función solo puede exportar un objeto, mientras que un módulo definido como clase puede exportar múltiples objetos.
- [ ] Un módulo definido como función puede tener múltiples instancias, mientras que un módulo definido como clase solo puede tener una.
- [ ] Un módulo definido como función no puede exportar atributos, mientras que un módulo definido como clase sí.

¿Cuál es el problema potencial al usar el primer enfoque para importar módulos en Node.js ("obtener el objeto/función allí donde sea requerido")? 1 punto

- [x] Puede dificultar el mantenimiento y la actualización, ya que los cambios en el módulo pueden requerir actualizaciones en muchos lugares diferentes.
- [ ] Puede mejorar el rendimiento porque reduce la cantidad de importaciones
- [ ] Puede causar errores de concurrencia porque cada importación crea una nueva instancia del módulo.
- [ ] Puede reducir la seguridad porque el módulo se comparte entre múltiples partes de la aplicación.

¿Cuál es una diferencia clave entre módulos definidos como objetos y módulos definidos como funciones en Node.js? 1 punto

- [x] Los módulos definidos como objetos siempre retornan la misma instancia cuando se importan, mientras que los definidos como funciones se ejecutan automáticamente.
- [ ] Los módulos definidos como objetos solo pueden exportar atributos, mientras que los definidos como funciones pueden exportar métodos.
- [ ] Los módulos definidos como objetos permiten crear múltiples instancias, mientras que los definidos como funciones no.
- [ ] Los módulos definidos como funciones no pueden acceder a variables globales, mientras que los definidos como objetos sí.

¿Qué beneficio ofrece la arquitectura modular en entornos de desarrollo ágiles y dinámicos? 1 punto

- [x] Facilita la implementación de cambios rápidos y la adaptación a requerimientos en constante cambio.
- [ ] Asegura la consistencia y estructura estricta, lo que mejora la mantenibilidad.
- [ ] Permite usar operaciones sincrónicas para evitar problemas de concurrencia.
- [ ] Facilita la creación de aplicaciones monolíticas para reducir la complejidad.

¿Cuál es la ventaja de usar módulos con múltiples exportaciones en una aplicación Node.js? 1 punto

- [x] Permite importar solo las partes necesarias de un módulo, reduciendo la carga y mejorando la modularidad.
- [ ] Siempre mejora el rendimiento porque reduce la cantidad de importaciones.
- [ ] Permite crear instancias únicas para cada exportación, mejorando la concurrencia.
- [ ] Reduce la necesidad de usar variables de aplicación (app.set() y app.get()), mejorando la seguridad.

¿Por qué algunas aplicaciones Node.js pueden prescindir de la capa de servicios? 1 punto

- [x] Porque la lógica de negocio es simple y puede implementarse directamente en los controladores, haciendo que la capa de servicios sea innecesaria.
- [ ] Porque la capa de servicios es solo para aplicaciones grandes y complejas.
- [ ] Porque los servicios son solo para aplicaciones que requieren acceso a bases de datos.
- [ ] Porque los servicios se usan solo para operaciones sincrónicas, mientras que Node.js es asíncrono por naturaleza.

¿Cuál de las siguientes afirmaciones describe la estructura modular de una aplicación pequeña y dinámica en Node.js? 1 punto

- [x] Puede prescindir de la capa de servicios, ya que la lógica de negocio se puede implementar directamente en los controladores o repositorios.
- [ ] Siempre requiere una capa de servicios, independientemente del tamaño de la aplicación.
- [ ] Solo necesita controladores y repositorios, sin necesidad de servicios ni entidades.
- [ ] Requiere una estructura monolítica para asegurar la consistencia y el rendimiento.

¿Qué ocurre si el siguiente código se ejecuta con un valor incorrecto para la variable name? 1 punto

```
module.exports = class Person {
  constructor(name) {
    this.name = name;
  }

  greeting() {
    return "Hello, " + this.name;
  }
};

const Person = require("./modules/person.js");

const person = new Person(undefined);
```

- (●) El método greeting() devolverá "Hello, undefined"
- ( ) Se lanzará un error, porque el constructor requiere un valor name no nulo.
- ( ) Se lanzará un error de referencia, porque this.name no está definido.
- ( ) El método greeting() devolverá "Hello, World!"

¿Qué ocurre si el siguiente código se ejecuta dos veces en una aplicación Node.js? 1 punto

```
const utils = require("./utils.js");

utils.someMethod();
```

- (●) La misma instancia del módulo utils se reutiliza en ambas ejecuciones, lo que puede compartir el estado entre importaciones
- ( ) Se crea una nueva instancia del módulo utils cada vez que se ejecuta, lo que puede causar inconsistencias.
- ( ) Se genera un error, porque someMethod() no está definido en utils
- ( ) Se reinicia el módulo, por lo que no se mantiene el estado entre ejecuciones.

¿Qué describe mejor la diferencia entre BSON y JSON en MongoDB? 1 punto

- (●) BSON es una versión binaria de JSON que permite almacenar datos en una estructura compacta.
- ( ) BSON es una versión ligera de JSON que se utiliza para representar documentos de bases de datos.
- ( ) BSON es un tipo de dato exclusivo de MongoDB, mientras que JSON se usa en todo el ecosistema JavaScript.
- ( ) BSON es más adecuado para datos sin estructura, mientras que JSON requiere esquemas fijos.

En MongoDB, ¿qué sucede cuando se intenta insertar un documento en una colección que no existe? 1 punto

- (●) Se crea automáticamente la colección y se inserta el documento.
- ( ) Se lanza un error indicando que la colección no existe.
- ( ) Se ignora el intento de inserción y el documento no se guarda.
- ( ) El documento se inserta en la colección predeterminada.

¿Cuál es un riesgo de usar MongoDB para aplicaciones que requieren transacciones ACID? 1 punto

- (●) MongoDB no es adecuado para transacciones ACID porque no garantiza Atomicity, Consistency, Isolation, ni Durability.
- ( ) MongoDB soporta transacciones ACID solo para documentos individuales.
- ( ) MongoDB no permite transacciones, ya que es una base de datos no relacional.
- ( ) MongoDB no es seguro para transacciones porque no soporta cifrado en reposo.

¿Qué hace el método updateOne en una colección de MongoDB? 1 punto

- (●) Actualiza el primer documento que coincide con el criterio de selección.
- ( ) Actualiza todos los documentos que coinciden con el criterio de selección.
- ( ) Elimina un documento de la colección.
- ( ) Inserta un nuevo documento si el criterio de selección no encuentra coincidencias.

¿Cuál es el propósito de ObjectID en MongoDB? 1 punto

- [x] Actuar como identificador único para cada documento en una colección.
- [ ] Representar el identificador del proceso que creó el documento.
- [ ] Garantizar que el documento es seguro para ser compartido en múltiples colecciones.
- [ ] Identificar el timestamp exacto en que se creó el documento.

¿Qué sucede si la URL de conexión a MongoDB es incorrecta al intentar conectarse? 1 punto

- [x] La función de callback recibirá un error indicando que la conexión falló.
- [ ] MongoDB intentará conectarse a la base de datos predeterminada.
- [ ] La conexión se realiza con las credenciales del usuario por defecto.
- [ ] Se lanza una excepción y se detiene la aplicación.

¿Cuál es el efecto de llamar a db.close() después de insertar un documento en MongoDB? 1 punto

- [x] Cierra la conexión a la base de datos, liberando recursos y evitando fugas de memoria.
- [ ] Se cierra el documento insertado, haciendo que no esté disponible para búsquedas futuras.
- [ ] Cierra solo la conexión del cliente actual, sin afectar a otras conexiones.
- [ ] No tiene efecto, porque MongoDB gestiona automáticamente las conexiones.

En MongoDB, ¿cuál es la diferencia entre insertOne() y insertMany()? 1 punto

- (●) insertOne() inserta un documento y insertMany() inserta varios documentos.
- ( ) insertOne() inserta un documento en una colección existente, mientras que insertMany() crea una nueva colección si no existe.
- ( ) insertOne() devuelve el documento insertado, mientras que insertMany() no devuelve nada.
- ( ) insertOne() es sincrónico, mientras que insertMany() es asíncrono.

Dado el siguiente código para insertar un documento, ¿qué sucede si la conexión a la base de datos falla? 1 punto

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/mydatabase', functi
  if (err) {
    console.error("Connection error: ", err);
  } else {
    const collection = db.collection('songs');
    const newSong = { title: 'Song 1', artist: 'Artist 1' };
    collection.insertOne(newSong, function(err, result) {
      if (err) {
        console.error("Insert error: ", err);
      } else {
        console.log("Document inserted with ID: ", result.inserted
      }
    });
  }
});
```

- (●) La función console.error("Connection error: ", err); se ejecutará, indicando un error de conexión.
- ( ) Se intentará reconectar automáticamente y luego insertar el documento.
- ( ) El documento se insertará en una colección por defecto.
- ( ) No sucederá nada, el error se ignorará.

En MongoDB, ¿qué sucede cuando se intenta encontrar un documento con un ObjectID incorrecto? 1 punto

- [x] No se encuentra ningún documento y no se lanza error.
- [ ] Se lanza un error, porque el ObjectID es inválido.
- [ ] El resultado es impredecible, depende del motor de base de datos.
- [ ] Se devuelve un documento por defecto.

Si usas updateOne() en MongoDB, ¿qué resultado esperas si el documento no existe y no se ha habilitado upsert? 1 punto

- [x] Ningún documento será actualizado y no habrá error.
- [ ] Se creará un nuevo documento.
- [ ] Se lanzará un error indicando que el documento no existe.
- [ ] Se actualizará el primer documento de la colección

Considera el siguiente código para insertar un documento en MongoDB usando promesas: ¿Qué sucede si el primer then() falla debido a una conexión incorrecta?

1 punto

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/mydatabase')
  .then(client => {
    const db = client.db('mydatabase');
    const collection = db.collection('songs');
    return collection.insertOne({ title: 'Song 2', artist: 'Artist 2' }
  })
  .then(result => {
    console.log("Document inserted with ID: ", result.insertedId);
  })
  .catch(err => {
    console.error("Error: ", err);
  });
```

- (●) El código en el bloque catch() se ejecutará, manejando el error.
- ( ) El segundo then() se ejecutará, pero sin insertar el documento
- ( ) Se intentará reconectar automáticamente.
- ( ) El documento se insertará en una colección por defecto.

¿Cuál es el propósito de usar async/await con operaciones de MongoDB?

1 punto

- (●) Facilitar la gestión de operaciones asíncronas, haciendo el código más legible.
- ( ) Asegurar que las operaciones se ejecuten de manera secuencial.
- ( ) Acelerar las operaciones de base de datos.
- ( ) Garantizar que las conexiones a la base de datos siempre sean exitosas.

¿Qué hace el siguiente código en MongoDB? 1 punto

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/mydatabase', (err, db) =
  if (err) {
    console.error("Connection error: ", err);
  } else {
    const collection = db.collection('songs');
    const query = { artist: 'Artist 1' };
    collection.find(query).toArray((err, docs) => {
      if (err) {
        console.error("Error fetching documents: ", err);
      } else {
        console.log("Documents: ", docs);
      }
    });
  }
});
```

- (●) Busca todos los documentos en la colección 'songs' donde el atributo 'artist' sea 'Artist 1'.
- ( ) Inserta un documento en la colección 'songs' con el atributo 'artist' igual a 'Artist 1'.
- ( ) Actualiza todos los documentos en la colección 'songs' para cambiar el 'artist' a 'Artist 1'.
- ( ) Elimina todos los documentos en la colección 'songs' donde 'artist' sea 'Artist 1'.

Si utilizas el módulo crypto para cifrar una cadena con createHmac, ¿qué sucede si la clave secreta utilizada para el cifrado es conocida por un atacante? 1 punto

- (●) El cifrado puede ser descifrado por el atacante utilizando la clave secreta conocida.
- ( ) El cifrado se vuelve vulnerable, pero no puede ser descifrado sin acceso al algoritmo exacto.
- ( ) La seguridad del cifrado permanece intacta, ya que el algoritmo es seguro por sí mismo.
- ( ) El cifrado será seguro siempre y cuando se utilicen valores de entrada aleatorios.

Si tienes un módulo que encapsula operaciones CRUD para una entidad llamada 'songs', ¿qué sucede si intentas insertar un documento sin un atributo requerido? 1 punto

- (●) Se lanzará un error y se llamará a callback(null).
- ( ) El documento se insertará, pero sin el atributo requerido.
- ( ) Se creará un documento con un atributo por defecto.
- ( ) La inserción será exitosa, pero el campo ausente se establecerá en undefined.

Si se desea agregar funcionalidades a la aplicación Express para procesar documentos BSON, ¿cuál de las siguientes opciones es la correcta? 1 punto

- (●) Importar el módulo body-parser y agregarlo a la aplicación con app.use(bodyParser.json()).
- ( ) Importar el módulo fs y leer archivos BSON directamente.
- ( ) Configurar el módulo MongoDB para procesar documentos BSON automáticamente.
- ( ) Convertir los documentos a JSON antes de enviarlos al cliente.

En el siguiente código, ¿cuál es el propósito del atributo resave en express-session? 1 punto

```
var expressSession = require('express-session');
app.use(expressSession({
  secret: 'abcdefg',
  resave: true,
  saveUninitialized: true
}));
```

- (●) Determina si la sesión debe ser guardada de nuevo incluso si no ha sido modificada durante la solicitud.
- ( ) Indica si la sesión debe ser creada automáticamente cuando un nuevo usuario se conecta.
- ( ) Define el tiempo de vida de la sesión antes de que se desconecte.
- ( ) Indica si la sesión debe ser destruida después de cada solicitud.

Si tienes una aplicación Express con sesiones implementadas utilizando express-session, ¿qué sucede si configuras saveUninitialized como false? 1 punto

- (●) La sesión solo se guardará si se ha realizado alguna modificación durante la solicitud.
- ( ) La sesión no se creará automáticamente cuando un nuevo usuario se conecte.
- ( ) La sesión se destruirá automáticamente si no hay actividad durante un periodo de tiempo.
- ( ) El valor por defecto del atributo de sesión se establecerá en undefined.

¿Cuál es la diferencia entre autenticación y autorización en el contexto de una aplicación Node.js? 1 punto

- (●) La autenticación verifica la identidad del usuario, mientras que la autorización verifica los permisos para acceder a recursos.
- ( ) La autenticación se usa para validar contraseñas, y la autorización se usa para crear tokens de acceso.
- ( ) La autenticación garantiza la seguridad de la aplicación, mientras que la autorización verifica la integridad de los datos.
- ( ) La autenticación es opcional, pero la autorización es obligatoria para todas las aplicaciones web.

En el siguiente código, ¿qué hace el controlador /logout? 1 punto

```
app.get('/logout', function (req, res) {
  req.session.user = null;
  res.send("Usuario desconectado");
});
```

- (●) Elimina la información del usuario de la sesión, efectivamente desconectándolo.
- ( ) Destruye completamente la sesión y redirige al usuario a la página de inicio.
- ( ) Redirecciona al usuario a la página de inicio sin destruir la sesión.
- ( ) Guarda la sesión actual pero elimina el atributo del usuario.

En un sistema de enrutadores en Node.js, ¿qué puede causar un error de orden de enrutadores y cómo puede evitarse? 1 punto

- [x] Los enrutadores deben añadirse antes de cualquier declaración app.get para que no se pierda la cadena de middleware.
- [ ] Los enrutadores deben añadirse al final de todas las declaraciones app.get para que no interfieran con otras rutas.
- [ ] Los enrutadores solo deben añadirse a rutas que no tengan controladores ya asignados para evitar conflictos.
- [ ] El orden de enrutadores no afecta al flujo de la aplicación, por lo que el orden no importa.

¿Cuál es el propósito de next() en un middleware de enrutador y qué sucede si no se llama a esta función en un enrutador? 1 punto

- [x] next() permite que la solicitud continúe a través de la cadena de middleware; si no se llama, la solicitud se detiene y el cliente recibe un error.
- [ ] next() obliga a la solicitud a avanzar al siguiente enrutador sin importar la lógica actual; si no se llama, la solicitud se reinicia.
- [ ] next() envía la solicitud al siguiente enrutador en la cadena de ejecución; si no se llama, la solicitud se cancela y se genera un error 404.
- [ ] next() indica al servidor que reintente la solicitud; si no se llama, el servidor intentará procesar la solicitud nuevamente.

¿Cuál es el efecto de usar un enrutador antes de un directorio de archivos estáticos? 1 punto

- [x] Permite que el enrutador realice lógica personalizada antes de servir archivos estáticos.
- [ ] Hace que todos los archivos estáticos sean inaccesibles, ya que el enrutador interceptará todas las solicitudes.
- [ ] Permite que el servidor ignore archivos estáticos a menos que el enrutador dé permiso explícito para acceder a ellos.
- [ ] No tiene efecto, ya que los enrutadores no afectan a archivos estáticos.

¿Qué sucede si defines un enrutador pero no llamas a next() ni realizas alguna operación que responda a la solicitud? 1 punto

- (●) La solicitud se detiene y el cliente queda a la espera de una respuesta que nunca llega.
- ( ) La solicitud se cancela automáticamente y el cliente recibe un error 404.
- ( ) El enrutador avanza automáticamente a la siguiente función en la cadena.
- ( ) El servidor termina la solicitud y redirige al cliente a la página de inicio.

¿Cómo puedes aplicar un enrutador para verificar si un usuario tiene una sesión activa antes de permitirle acceso a ciertas rutas? 1 punto

- (●) Definiendo un enrutador que compruebe si req.session.user está definido y, si no es así, redireccionar al usuario a la página de inicio de sesión.
- ( ) Definiendo un enrutador que destruya la sesión si el usuario no está autenticado y redireccionar a la página de error.
- ( ) Configurando el enrutador para reiniciar la sesión si el usuario no tiene una sesión activa.
- ( ) El enrutador no puede controlar la autenticación; debes manejarlo dentro de cada controlador.

¿Qué sucede si intentas procesar un formulario con un campo de tipo file, pero no configuras el enctype correctamente? 1 punto

- (●) El servidor recibirá los datos como texto en lugar de archivos binarios, lo que puede causar errores al intentar acceder a los archivos.
- ( ) El servidor ignorará el campo de tipo file y procederá con el resto de la solicitud, como si el campo de archivo no estuviera presente.
- ( ) El servidor generará un error 500 indicando un problema con la solicitud HTTP.
- ( ) La aplicación expresará un error de validación indicando que el tipo de contenido del formulario es incorrecto.

¿Cuál es el comportamiento del método file.mv() si el directorio de destino no existe? 1 punto

- (●) El método generará un error porque no puede mover el archivo a un directorio inexistente.
- ( ) El método creará automáticamente el directorio y moverá el archivo sin problemas.
- ( ) El método intentará crear el directorio, pero si no tiene permisos suficientes, generará un error.
- ( ) El método moverá el archivo a un directorio temporal si el directorio de destino no existe.

¿Cuál es una posible vulnerabilidad al aceptar archivos desde el cliente y cómo puedes mitigarla? 1 punto

- (●) El riesgo de inyección de archivos maliciosos; puedes mitigarla limitando el tipo de archivo aceptado y validando el contenido antes de procesarlo.
- ( ) El riesgo de que los usuarios envíen archivos de gran tamaño; puedes mitigarla limitando el tamaño máximo de los archivos subidos.
- ( ) El riesgo de duplicación de archivos; puedes mitigarla utilizando nombres únicos para cada archivo subido.
- ( ) El riesgo de perder archivos subidos; puedes mitigarla haciendo copias de seguridad automáticas de los archivos subidos.

¿Cuál es la consecuencia de no cerrar la conexión de archivo después de llamar a file.mv()? 1 punto

- (●) Puede resultar en fugas de memoria y otros problemas de rendimiento, ya que la conexión de archivo permanece abierta.
- ( ) No cerrar la conexión de archivo no afecta el rendimiento ni la estabilidad de la aplicación.
- ( ) La conexión de archivo se cerrará automáticamente cuando el servidor se reinicie.
- ( ) No cerrar la conexión de archivo provoca errores de acceso a datos, ya que las conexiones abiertas se acumulan y saturan el sistema.

¿Cómo puedes limitar el acceso a archivos subidos para mejorar la seguridad en una aplicación Node.js? 1 punto

- [x] Puedes restringir el acceso a ciertos directorios solo para usuarios autenticados y verificar los permisos de acceso antes de permitir el acceso a los archivos.
- [ ] Puedes implementar medidas de cifrado para archivos confidenciales y solo descifrarlos cuando se necesiten.
- [ ] Puedes establecer reglas en el servidor para evitar el acceso no autorizado a los archivos subidos.
- [ ] Puedes mover todos los archivos subidos a un servidor seguro y gestionar el acceso desde allí.

¿Qué sucede si aplicas un valor incorrecto al método skip() en MongoDB para la paginación? 1 punto

- [ ] MongoDB ignorará el valor incorrecto y procederá con el siguiente documento en la colección.
- [x] MongoDB lanzará un error que detendrá la ejecución de la consulta.
- [ ] MongoDB arrojará un valor predeterminado y continuará ejecutando la consulta.
- [ ] MongoDB arrojará todos los documentos de la colección sin paginación.

¿Qué sucede si olvidas aplicar el método limit() al realizar paginación en MongoDB? 1 punto

- [x] La consulta devolverá todos los documentos de la colección, ignorando el número de página.
- [ ] La consulta lanzará un error indicando que falta el parámetro limit.
- [ ] La consulta devolverá solo el primer documento de la colección.
- [ ] La consulta devolverá solo los últimos 10 documentos de la colección.

¿Cuál es el propósito del campo pg en una URL para paginación en Node.js? 1 punto

- (•) Indicar el número de página a mostrar en la consulta.
- ( ) Especificar el número de documentos por página.
- ( ) Especificar el criterio de orden para la consulta.
- ( ) Especificar el número máximo de páginas a mostrar.

En el siguiente código, ¿cuál sería el resultado si el parámetro pg es null o no es un número válido? 1 punto

```
app.get("/ads", function(req, res) {
  var pg = parseInt(req.query.pg);
  if (isNaN(pg)) {
    pg = 1;
  }
  obtenerAnuncios(pg, function(anuncios) {
    res.send(anuncios);
  });
});
```

- (•) Se asigna 1 a pg y se obtienen los anuncios de la primera página.
- ( ) Se lanza un error, ya que pg debe ser un número entero.
- ( ) Se asigna null a pg y la consulta se ejecuta sin paginación.
- ( ) Se asigna un valor predeterminado y se ejecuta la consulta sin errores.

¿Qué sucede si utilizas el mismo valor para skip() y limit() en una consulta paginada en MongoDB? 1 punto

```
collection.find({}).skip(10).limit(10);
```

- (●) La consulta devolverá 10 documentos comenzando desde el décimo documento en adelante.
- ( ) La consulta devolverá los primeros 10 documentos de la colección.
- ( ) La consulta devolverá solo el décimo documento.
- ( ) La consulta lanzará un error debido a un conflicto entre skip() y limit().

¿Qué ocurre si no se captura un error de MongoDB en una aplicación Node.js? 1 punto

- (●) El error se propagará hasta la capa superior y provocará la finalización inesperada de la aplicación.
- ( ) El error será ignorado, y la aplicación continuará funcionando normalmente
- ( ) El error se capturará automáticamente y se mostrará un mensaje genérico al usuario.
- ( ) La aplicación generará un nuevo error con un mensaje más detallado.

¿Cuál es el propósito de incluir una función de manejo de errores global con app.use() al final de la aplicación Node.js? 1 punto

- (●) Para capturar todos los errores no controlados y evitar que se muestren detalles sensibles.
- ( ) Para capturar errores solo durante el desarrollo y proporcionar trazas detalladas.
- ( ) Para mostrar mensajes personalizados a los usuarios cuando se produce un error.
- ( ) Para reiniciar automáticamente la aplicación cuando se produce un error crítico.

¿Qué sucede si intentas enviar una respuesta HTTP después de que se hayan enviado los encabezados en Node.js? 1 punto

- (●) Se lanza un error porque no se pueden enviar múltiples respuestas a la misma solicitud.
- ( ) Se crea una nueva respuesta con encabezados duplicados.
- ( ) Se sobrescribe la respuesta original y se envía la nueva respuesta.
- ( ) La respuesta original se envía correctamente, pero la segunda respuesta se ignora.

En el siguiente fragmento de código, ¿por qué es importante verificar si res.headersSent antes de enviar una respuesta? 1 punto

```
app.use(function(err, req, res, next) {
  console.log("Error producido: " + err);
  if (!res.headersSent) {
    res.send("Recurso no disponible");
  }
});
```

- (●) Para evitar errores al intentar enviar una respuesta después de que se hayan enviado los encabezados.
- ( ) Para asegurarse de que no se envíen múltiples respuestas a la misma solicitud.
- ( ) Para evitar la duplicación de datos en la respuesta.
- ( ) Para permitir la reutilización de la conexión HTTP.

En Node.js, ¿cuál es el resultado de no capturar un error en una función asincrónica? 1 punto

- La ejecución de la función se detiene, y el flujo de control se transfiere al siguiente bloque de código.
- El proceso de Node.js se detiene inmediatamente y muestra un error crítico.
- El error se propaga hacia arriba, lo que puede hacer que la aplicación se cierre si no se controla.
- El error se ignora, y la función continúa su ejecución.

¿Por qué es importante capturar errores en la conexión a una base de datos MongoDB en Node.js? 1 punto

- Para evitar que la aplicación se bloquee y proporcionar información útil para la depuración.
- Para asegurarse de que todos los datos se guardan correctamente en la base de datos.
- Para permitir el reinicio automático de la conexión a la base de datos
- Para evitar que la conexión se realice dos veces.

¿Qué sucede si intentas enviar una respuesta HTTP en Node.js después de que la conexión haya sido cerrada por el cliente? 1 punto

- Se genera un error porque la conexión ya no está activa.
- Node.js intentará abrir una nueva conexión para enviar la respuesta.
- La respuesta se almacena en caché hasta que la conexión se reabra.
- El servidor ignora la solicitud de respuesta porque ya no puede enviarla.

Si estás utilizando HTTPS en una aplicación Node.js, ¿por qué es importante verificar que el certificado sea emitido por una autoridad de certificación confiable? 1 punto

- [x] Porque los navegadores solo confiarán en certificados emitidos por autoridades confiables.
- [ ] Porque los certificados no confiables pueden ser rechazados por el servidor.
- [ ] Porque Node.js requiere certificados confiables para funcionar con HTTPS.
- [ ] Porque las autoridades confiables garantizan la seguridad de la aplicación.