TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Uso del paquete body-parser.
- Códigos de Respuesta HTTP.
- Los distintos códigos de respuesta del protocolo HTTP.
- Códigos personalizados.

USO DEL PAQUETE BODY-PARSER

El `body-parser` analiza el cuerpo de las solicitudes entrantes en un middleware, antes que sus controladores disponibles en la propiedad `req.body`.

Este módulo proporciona los siguientes analizadores:

- Analizador de cuerpo JSON.
- Analizador de cuerpo sin procesar.
- Analizador de cuerpo de texto.
- Analizador de cuerpo de formulario con codificación URL.

Por ejemplo:

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.json());

app.post('/', (req, res) => {
    res.json(req.body);
});

app.listen(3000, () => console.log('server started'));
```

`bodyParser.json()` analiza la cadena del cuerpo de la solicitud JSON en un objeto de JavaScript, y luego lo asigna al objeto `req.body`.

Establezca el encabezado `Content-Type` en la respuesta a `application/json; charset=utf-8`, sin ningún cambio. El método anterior se aplica a la mayoría de los otros framework en back-end.

CÓDIGOS DE RESPUESTA HTTP

Estos estandarizan una forma de informar al cliente sobre el resultado de su solicitud.

Para eliminar la confusión de los usuarios de la API cuando ocurre un error, debemos manejarlos sutilmente, y devolver códigos de respuesta HTTP que indiquen qué tipo de error ocurrió, brindando a sus mantenedores suficiente información para comprenderlo. La idea principal es que no se desea que los errores debiliten nuestro sistema, por lo que podemos dejarlos sin manejar, pero esto acarrea que el cliente o consumidor de la API tiene que manejarlos.

Deberíamos arrojar errores que correspondan al problema que ha encontrado nuestra aplicación. Por ejemplo, si queremos rechazar los datos de la carga útil de la solicitud, se devuelve una respuesta 400 de la siguiente manera en una API Express:

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

// existing users
const users = [{
    email: 'abcdefg@test.com'
}]

app.use(bodyParser.json());

app.post('/users', (req, res) => {
    const {
        email
    } = req.body;
    const userExists = users.find(u => u.email === email);
    if (userExists) {
        return res.status(400).json({
            error: 'El usuario ya existe'
        })
    }
    res.json(req.body);
});

app.listen(3000, () => console.log('server started'));
```

En el anterior código se observa una lista de usuarios existentes en la matriz de usuarios, con el campo dado de correo electrónico.

Al intentar enviar la carga útil en el campo con un valor de correo electrónico existente de usuarios, obtendremos un código de estado de respuesta 400 con el mensaje "El usuario ya existe", con la finalidad de que el cliente o el consumidor de la API capture dicha alerta. Con esa información, el cliente puede corregir la acción cambiando el correo electrónico por uno que no esté registrado.

Los códigos de error deben tener mensajes que los acompañen, para que así los mantenedores tengan suficiente información para solucionar el problema; pero los atacantes no pueden usar el contenido del error para llevar a cabo nuestros ataques, como el robo de información o la caída del sistema.

Cada vez que nuestra API no se complete con éxito, deberíamos fallar correctamente, enviando un error con información para ayudar a los usuarios a realizar acciones correctivas.

INSPECCIONAR LOS DISTINTOS CÓDIGOS DE RESPUESTA DEL PROTOCOLO HTTP: 1XX, 2XX, 3XX, 4XX, 5XX

Ahora, se mostrarán algunos códigos de respuesta que generalmente veremos al realizar pruebas de API REST en POSTMAN, o en cualquier cliente API REST.

Serie 100	**Estas son respuestas temporales.**
100	Continuar
101	Protocolos de conmutación
102	Procesamiento

Serie 200	**El cliente acepta la solicitud, siendo procesada con éxito en el servidor.**
200	Ok

201	Creado
202	Aceptado
203	Información no autorizada
204	Sin contenido
205	Restablecer contenido
206	Contenido parcial
207	Estado múltiple
208	Ya informado
226	IM usado

Serie 300	**Relacionados con esta serie son para redireccionamiento de URL.**
300	Varias opciones
301	Movido permanentemente
302	Encontrado
303	Marque otro
304	No modificado
305	Usar proxy
306	Cambiar proxy

307	Re-direccionamiento temporal
308	Re-direccionamiento permanente

Serie 400	**Errores del lado del cliente.**
400	Petición Incorrecta
401	No autorizado
402	Pago requerido
403	Prohibido
404	No encontrado
405	Método no permitido
406	No aceptable
407	Se requiere autenticación de proxy
408	Solicitar tiempo de espera
409	Conflicto
410	Ido
411	Longitud requerida
412	Condición previa fallida
413	Carga útil demasiado grande

414	URI demasiado largo
415	Tipo de papel no admitido
416	Rango no satisfactorio
417	Expectativa fallida
418	Soy una tetera
421	Solicitud mal dirigida
422	Entidad no procesable
423	Bloqueado
424	Dependencia fallida
426	Requiere actualización
428	Requisito previo
429	Demasiadas solicitudes
431	Campos de encabezado de solicitud demasiado grandes
451	No disponible por motivos legales

Serie 500	**Errores del lado del servidor.**
500	Error interno de servidor
501	No implementado

502	Puerta de enlace no válida
503	Servicio no Disponible
504	Tiempo de espera de la puerta de enlace
505	Versión HTTP no compatible
506	La variante también negocia
507	Almacenamiento insuficiente
508	Bucle detectado
510	No extendido
511	Se requiere autenticación de red

RESPONDIENDO CÓDIGOS PERSONALIZADOS EN UNA PETICIÓN HTTP

Cuando se maneja una petición de API RESTful, si ocurre un error en la petición del usuario, o si algo inesperado ocurre en el servidor, simplemente se puede lanzar una excepción para notificar al usuario que algo erróneo ocurrió. Si logra identificar la causa del error (por ejemplo: el recurso solicitado no existe), se debe considerar lanzar una excepción con el código HTTP de estado apropiado.

Algunas veces se puede personalizar el formato de la respuesta del error por defecto. Por ejemplo, en lugar de depender del uso de diferentes estados HTTP para indicar los errores, se puede usar siempre el estado HTTP 200, y encapsular su código real como parte de una estructura JSON en la respuesta, tal como se muestra a continuación:

```
HTTP / 1.1 200 OK
Date: Sun, 02 Mar 2014 05: 31: 43 GMT
Server: Apache / 2.2 .26(Unix) DAV / 2 PHP / 5.4 .20 mod_ssl / 2.2 .26
OpenSSL / 0.9 .8 y
Transfer - Encoding: chunked
```

```
Content - Type: application / json;
charset = UTF - 8
```

```
{
   "success":false,
   "data":{
      "name":"Not Found Exception",
      "message":"The requested resource was not found.",
      "code":0,
      "status":404
   }
}
```

En Express se puede configurar el estado de una respuesta de la la siguiente manera:

```
res.status(500).send({
    error: 'Error interno del Servidor'
})
```

Por ejemplo, observemos el siguiente código:

```
const express = require("express");
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.urlencoded({
    extended: false
}));
app.use(bodyParser.json());

let usuario = {
    nombre: '',
    apellido: ''
};

let respuesta = {
    error: false,
    codigo: 200,
    mensaje: ''
};

app.post('/usuario', function(req, res) {
    if (!req.body.nombre || !req.body.apellido) {
        respuesta = {
            error: true,
            codigo: 502,
            mensaje: 'El campo nombre y apellido son requeridos'
        };
    } else {
        if (usuario.nombre === '' || usuario.apellido === '') {
            respuesta = {
                error: true,
```

```
                codigo: 503,
                mensaje: 'El campo nombre o apellido no puede ser vacío'
            };
        } else {
            usuario = {
                nombre: req.body.nombre,
                apellido: req.body.apellido
            };
            respuesta = {
                error: false,
                codigo: 200,
                mensaje: 'Usuario creado',
                respuesta: usuario
            };
        }
    }
    res.send(respuesta);
});

app.use(function(req, res, next) {
    respuesta = {
        error: true,
        codigo: 400,
        mensaje: 'URL no encontrada'
    };
    res.status(404).send(respuesta);
});
app.listen(3000, () => {
    console.log("El servidor está inicializado en el puerto 3000");
});
```

Observamos que primero se inicia Express, pero ahora también cargamos el body-parser para utilizar Express en nuestra constante app. Seguidamente a body-parser, el cual permite manejar datos en formato JSON de la recepción en POST/PUT desde el cliente, es decir, cuando el cliente haga una petición, se indicará la cabecera `Content-Type: application/json`, y la API la realizará ejecutando lo que el cliente necesita. Tenemos un código de **respuesta** personalizado llamado "respuesta", que está conformado por tres campos: error, código y el mensaje, que nos permitirá dar respuesta al cliente.

En la ruta para crear un usuario haciendo uso del `POST`, vemos varias respuestas que se muestran al cliente. La primera es que si algunos de los dos campos nombre y apellidos no se incluyen, se refleja un error personalizado al cliente de 502, indicando que ambos son requeridos. Luego, si incluye ambos campos pero alguno de los dos, o los dos, están vacíos, refleja al cliente un error 501 indicando que no se ha creado el mismo. Finalmente, si posee los dos campos no vacíos, se notifica que el Usuario ha sido creado con un mensaje de error en falso y código 200, respectivamente.