Implementación de middleware de manejo de errores personalizados en una aplicación Express

Esfuerzo Estimado: 30 mins

Un manejo adecuado de errores hace que el código sea más mantenible y facilita la depuración. Ayuda a los desarrolladores a identificar y solucionar problemas durante las fases de desarrollo y mantenimiento.

Objetivos

En este laboratorio aprenderás a:

1. Crear Errores Personalizados
2. Manejar errores personalizados con un manejador de errores global

Requisitos Previos

Debes haber realizado previamente las tareas de manejo de errores incluidas en este laboratorio.

Iniciar servidor mongodb

Notarás que MongoDB está listado allí, pero inactivo. Lo que significa que la base de datos no está disponible para usar.

Una vez que hagas clic en él, verás más detalles sobre el mismo y un botón para iniciarlo.

Hacer clic en el botón Crear ejecutará un proceso en segundo plano para configurar e iniciar tu servidor MongoDB.

Poco después de eso, tu servidor estará listo para usar. Este despliegue tiene el control de acceso habilitado y MongoDB aplica la autenticación. Así que, **toma nota de la contraseña** ya que la necesitarás para iniciar sesión como usuario `root`.
`Note:` Para la contraseña y otra información haz clic en `Connection Information`

Crear Aplicación Node

Crearás una aplicación e incluirás pasos para crear un manejo de errores personalizado. Utiliza el mismo repositorio de código que se usó para `CustomerPortal`, que incluía registro de usuarios e inicio de sesión. Agregarás manejo de errores a esa aplicación.

1. Abre un nuevo terminal.

2. Clona el repositorio `https://github.com/ibm-developer-skills-network/abltc-backend-nodejs-customerportal.git` al entorno del laboratorio ejecutando el siguiente comando.

   ```
   git clone https://github.com/ibm-developer-skills-network/abltc-backend-nodejs-customerportal.git
   ```

3. Cambia al directorio recién creado ejecutando el siguiente comando.

```
cd abltc-backend-nodejs-customerportal
```

4. Ejecuta el siguiente comando en la terminal para instalar todos los paquetes que son necesarios.

```
npm install
```

Todos los paquetes necesarios ya han sido configurados en package.json.

5. Abre el archivo `customer_app.js` en el editor. Reemplaza la contraseña de marcador de posición con la contraseña real asignada por tu instancia del servidor MongoDB.

6. Vamos a añadir el manejador de errores global a este código. Justo antes de iniciar la aplicación, incluye el siguiente código.

```
app.use((err,req,res,next) => {
    err.statusCode = err.statusCode || 500;
    err.status = err.status || "Error";
    console.log(err.stack);
    res.status(err.statusCode).json({
        status: err.statusCode,
        message: err.message,
    });
})
```

7. Agrega el siguiente middleware para manejar todas las solicitudes inválidas que lleguen a la aplicación.

```
app.all("*",(req,res,next)=>{
    const err = new Error(`Cannot find the URL ${req.originalUrl} in this application. Please check.`);
    err.status = "Endpoint Failure";
    err.statusCode = 404;
    next(err);
})
```

7. Ejecuta el customer_app.js desde la terminal.

```
node customer_app.js
```

8. Haz clic en el botón de abajo para iniciar la aplicación. Añade `checkIfEsists` a la URL en el navegador para intentar acceder a un punto final que no existe y ver cómo se maneja.

Customer Portal

▼ Haz clic aquí si el botón de arriba no funciona

1. Haz clic en el ícono de `Skills Network` en la barra de herramientas izquierda.
2. Elige `Launch Application`.
3. Ingresa el número de puerto `3000` en el que se está ejecutando la aplicación.
4. Haz clic en el ícono, como se muestra en la imagen, para abrir el portal del cliente en una nueva pestaña.

Verás que la aplicación no se bloquea, sino que devuelve un objeto y puedes continuar accediendo a otros puntos finales.

9. Presiona `Ctrl+c` y detén el servidor.

Crear Errores Personalizados

Si un usuario intenta crear un cliente con datos inválidos, puedes usar middleware de manejo de errores personalizado para validar el cuerpo de la solicitud y enviar una respuesta de error apropiada. De manera similar, si hay un error al guardar los datos del cliente en la base de datos, puedes manejarlo utilizando middleware de manejo de errores personalizado y proporcionar un mensaje de error informativo.

En el caso de UserRegistration, tendrás los siguientes errores.

- Edad del cliente < 21
- El nombre de usuario ya existe
- Intento de inicio de sesión para un usuario inexistente
- Intento de inicio de sesión con una contraseña incorrecta

Ahora crearás controladores personalizados para este propósito.

1. Crea una nueva carpeta llamada `errors` utilizando el comando proporcionado y asegúrate de navegar al directorio de errores.

   ```
   mkdir errors
   cd errors
   ```

2. Crea un nuevo archivo llamado `CustomError.js` ejecutando el siguiente comando.

   ```
   touch CustomError.js
   ```

3. Abre el archivo `CustomError.js` en el editor y pega el siguiente contenido.
   Estarás creando una nueva clase de error llamada `ValidationError` que extiende la clase `Error`.

   ```
   /**
    * Custom Error Class: ValidationError
    *
    * Represents an error that occurs when validation fails.
    * Inherits from the built-in Error class.
    */
   class ValidationError extends Error {
       constructor(message) {
           // Call the constructor of the parent class (Error)
           super(message);
           // Custom properties for the ValidationError
           this.code = 407; // Custom error code
           this.name = "ValidationError"; // Custom error name
       }
   }
   ```

```
/**
 * Custom Error Class: InvalidUserError
 *
 * Represents an error that occurs when dealing with an invalid user.
 * Inherits from the built-in Error class.
 */
class InvalidUserError extends Error {
    constructor(message) {
        // Call the constructor of the parent class (Error)
        super(message);
        // Custom properties for the InvalidUserError
        this.code = 407; // Custom error code
        this.name = "InvalidUserError"; // Custom error name
    }
}
/**
 * Custom Error Class: AuthenticationFailed
 *
 * Represents an error that occurs when authentication fails.
 * Inherits from the built-in Error class.
 */
class AuthenticationFailed extends Error {
    constructor(message) {
        // Call the constructor of the parent class (Error)
        super(message);
        // Custom properties for the AuthenticationFailed error
        this.code = 407; // Custom error code
        this.name = "AuthenticationFailed"; // Custom error name
    }
}
module.exports = { ValidationError, InvalidUserError, AuthenticationFailed};
```

- ValidationError se crea y se lanza cuando los datos proporcionados para el registro de usuario no son aceptables, como se explicó anteriormente: edad menor de 21 años o si el nombre de usuario ya existe en la base de datos.
- InvalidUserError se crea y se lanza cuando el nombre de usuario es inválido durante el intento de inicio de sesión.
- AuthenticationFailed también es un error, aunque no tiene la palabra `Error` ya que hereda de la clase Error. Se lanza cuando la contraseña es inválida durante un intento de inicio de sesión.

Lanzar los Errores Personalizados en el Registro y Login

1. Cambia el directorio a `abltc-backend-nodejs-customerportal` ejecutando el siguiente comando.

   ```
   cd ..
   ```

2. Abre `customer_app.js` con el editor y requiere `errors/CustomErrors.js` dentro de él.

```
const { ValidationError, InvalidUserError, AuthenticationFailed } = require('./errors/CustomError');
```

3. Reemplace el código `/api/add_customer` con el siguiente código.

```
// POST endpoint for adding a new customer
app.post('/api/add_customer', async (req, res, next) => {
    const data = req.body;
    const age = parseInt(data['age']);

    try {
        if (age < 21) {
            throw new ValidationError("Customer Under required age limit");
        }

        const customer = new Customers({
            "user_name": data['user_name'],
            "age": age,
            "password": data['password'],
            "email": data['email']
        });

        await customer.save();

        res.send("Customer added successfully");
    } catch (error) {
        next(error);
    }
});
```

4. Reemplace el código `/api/login` con el siguiente código.

```
// POST endpoint for user login
app.post('/api/login', async (req, res, next) => {
    const data = req.body;
    const user_name = data['user_name'];
    const password = data['password'];

    try {
        const user = await Customers.findOne({ user_name: user_name });
        if (!user) {
            throw new InvalidUserError("No such user in database");
        }
        if (user.password !== password) {
            throw new AuthenticationFailed("Passwords don't match");
        }
        res.send("User Logged In");
    } catch (error) {
        next(error);
    }
});
```

5. Agrega el siguiente código al archivo para manejar `logout`.

```
// GET endpoint for user logout
app.get('/api/logout', async (req, res) => {
    res.cookie('username', '', { expires: new Date(0) });
    res.redirect('/');
});
```

6. Ahora ejecuta el siguiente comando y arranca el servidor.

```
node customer_app.js
```

Esto inicia el servidor express en el puerto `3000`.

7. Haz clic en el botón de abajo para lanzar la aplicación.

Customer Portal

▶ Haz clic aquí si el botón de arriba no funciona

8. Regístrate con una edad menor de 21 años y verifica si se muestra un mensaje de error apropiado. Deberías recibir una salida como la siguiente.

```
{"status":500,"message":"Customer Under required age limit"}
```

También verás la pila de errores impresa en la terminal.

9. Regístrate con una edad mayor o igual a 21. Se registrará con éxito y añadirá al usuario a la base de datos.

10. Inicia sesión con un nombre de usuario que no existe. Verifica si recibes un mensaje de error apropiado. Deberías recibir una salida como la siguiente.

    ```
    {"status":500,"message":"No such user in database"}
    ```

También verás la pila de errores impresa en la terminal.

11. Inicia sesión con el nombre de usuario correcto y la contraseña incorrecta. Verifica si recibes un mensaje de error apropiado. Deberías recibir una salida como la siguiente.

    ```
    {"status":500,"message":"Passwords don't match"}
    ```

También verás la pila de errores impresa en la terminal.

Ejercicio Práctico

1. Agrega el campo `Name` a la colección de Clientes e implementa un manejo de errores personalizado para validar que `Name` sea solo una cadena.
2. Implementa un manejo de errores personalizado para el Registro de Usuarios con brcrypt y JWT.

Author(s)

Lavanya T S