Global Error Handling

Global error handling in Node.js is essential for ensuring that your application gracefully handles unexpected errors and prevents crashes. There are multiple ways to implement global error handling, depending on whether you're dealing with synchronous, asynchronous, or unhandled errors.

1. Express Global Error Handling Middleware

In an Express application, you can define a centralized error-handling middleware.

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

// Middleware to simulate an error
app.get('/', (req, res, next) => {
    throw new Error('Something went wrong!');
});

// Global Error Handling Middleware
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({ message: err.message });
});

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

Handling Asynchronous Errors with `next()`

If you're using asynchronous code, pass errors to `next()` so Express can catch them.

```
app.get('/async-error', async (req, res, next) => {
    try {
        await Promise.reject(new Error('Async error occurred!'));
    } catch (err) {
        next(err);
    }
});
```

2. Using a Centralized Error Handler Class

Creating a centralized error handler helps organize error management.

```
class AppError extends Error {
    constructor(message, statusCode) {
        super(message);
```

```
        this.statusCode = statusCode;
        this.isOperational = true;

        Error.captureStackTrace(this, this.constructor);
    }
}

// Middleware to handle application-specific errors
app.use((err, req, res, next) => {
    if (err instanceof AppError) {
        return res.status(err.statusCode).json({ message: err.message });
    }
    console.error(err);
    res.status(500).json({ message: 'Internal Server Error' });
});

// Throwing a custom error
app.get('/custom-error', (req, res, next) => {
    throw new AppError('Custom error occurred!', 400);
});



const express = require("express");

const app = express();
app.use(express.json()); // Middleware to parse JSON request bodies

// ==========================
// Custom Error Handler Class
// ==========================
class AppError extends Error {
    constructor(message, statusCode) {
        super(message);
        this.statusCode = statusCode;
        this.isOperational = true; // Indicates a known, expected error
        Error.captureStackTrace(this, this.constructor);
    }
}

// ==========================
// Sample Routes
// ==========================

// 1 Normal Route
app.get("/", (req, res) => {
    res.send("Welcome to the Express app!");
});

// 2 Synchronous Error (Handled by Global Error Middleware)
app.get("/sync-error", (req, res, next) => {
    throw new AppError("Synchronous error occurred!", 400);
});

// 3 Asynchronous Error (Handled by Global Error Middleware)
app.get("/async-error", async (req, res, next) => {
    try {
        await Promise.reject(new AppError("Asynchronous error occurred!",
500));
    } catch (err) {
        next(err); // Pass error to error handling middleware
```

```
    }
});

// ========================
// Global Error Handling Middleware
// ========================
app.use((err, req, res, next) => {
    console.error("Error:", err.message);

    // If it's an instance of AppError, return a structured error response
    if (err instanceof AppError) {
        return res.status(err.statusCode).json({
            status: "error",
            message: err.message,
        });
    }

    // Handle unknown errors (Internal Server Errors)
    res.status(500).json({
        status: "error",
        message: "Something went wrong!",
    });
});

// ========================
// Start Server
// ========================
const PORT = 3000;
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

How It Works

Endpoint	Description
`GET /`	Normal route (returns welcome message)
`GET /sync-error`	Throws a **synchronous error** (`AppError` with 400 status)
`GET /async-error`	Throws an **asynchronous error** (`AppError` with 500 status)

Expected Responses

Normal Route (`GET /`)

```
Welcome to the Express app!
```

Synchronous Error (`GET /sync-error`)

```
{
    "status": "error",
    "message": "Synchronous error occurred!"
}
```

Asynchronous Error (`GET /async-error`)

```
{
```

```
    "status": "error",
    "message": "Asynchronous error occurred!"
}
```

Key Benefits of This Approach

Centralized Error Handling → Keeps code clean and maintainable.
Handles Both Sync & Async Errors → Uses `next(err)` for async cases.
Proper Error Response Format → Returns structured JSON responses.
Extensible → You can add more error-handling logic like logging with Winston.

Centralized error handling system in a Node.js Express application with a structured architecture.

Project Structure

```
/error-handler-app
├── /src
│   ├── /controllers
│   │   ├── user.controller.ts      # Controller layer
│   ├── /services
│   │   ├── user.service.ts         # Service layer
│   ├── /middlewares
│   │   ├── error.middleware.ts     # Global error-handling middleware
│   ├── /utils
│   │   ├── appError.ts             # Custom error class
│   ├── app.ts                      # Main Express app
├── server.ts                        # Entry point
├── tsconfig.json
├── package.json
```

1 Custom Error Class (`appError.ts`)

Located in: **`/src/utils/appError.ts`**
This class creates structured error messages.

```
export class AppError extends Error {
    public statusCode: number;
    public isOperational: boolean;

    constructor(message: string, statusCode: number) {
        super(message);
        this.statusCode = statusCode;
        this.isOperational = true;
        Error.captureStackTrace(this, this.constructor);
    }
```

```
}
```

2 Global Error Handling Middleware (`error.middleware.ts`)

Located in: **`/src/middlewares/error.middleware.ts`**
Handles all errors in one place.

```
import { Request, Response, NextFunction } from "express";
import { AppError } from "../utils/appError";

export const errorMiddleware = (
    err: Error,
    req: Request,
    res: Response,
    next: NextFunction
) => {
    console.error("Error:", err.message);

    if (err instanceof AppError) {
        return res.status(err.statusCode).json({
            status: "error",
            message: err.message,
        });
    }

    res.status(500).json({
        status: "error",
        message: "Something went wrong!",
    });
};
```

3 Service Layer (`user.service.ts`)

Located in: **`/src/services/user.service.ts`**
Contains business logic.

```
import { AppError } from "../utils/appError";

export const getUser = (id: string) => {
    if (id !== "1") {
        throw new AppError("User not found!", 404);
    }
    return { id: 1, name: "John Doe" };
};
```

4 Controller Layer (`user.controller.ts`)

Located in: **`/src/controllers/user.controller.ts`**
Handles HTTP requests and calls services.

```
import { Request, Response, NextFunction } from "express";
```

```
import { getUser } from "../services/user.service";
import { AppError } from "../utils/appError";

export const getUserController = (req: Request, res: Response, next:
NextFunction) => {
    try {
        const userId = req.params.id;
        const user = getUser(userId);
        res.status(200).json(user);
    } catch (err) {
        next(err);
    }
};

// Async error example
export const asyncErrorController = async (req: Request, res: Response,
next: NextFunction) => {
    try {
        await Promise.reject(new AppError("Async error occurred!", 500));
    } catch (err) {
        next(err);
    }
};
```

5 Express App (`app.ts`)

Located in: **`/src/app.ts`**
Sets up routes and error handling.

```
import express from "express";
import { getUserController, asyncErrorController } from
"./controllers/user.controller";
import { errorMiddleware } from "./middlewares/error.middleware";

const app = express();
app.use(express.json());

// Routes
app.get("/user/:id", getUserController);
app.get("/async-error", asyncErrorController);

// Global Error Handler
app.use(errorMiddleware);

export default app;
```

6 Server (`server.ts`)

Located in: **`server.ts`**
Starts the Express server.

```
import app from "./src/app";

const PORT = 3000;
```

```
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

Test the API Endpoints

Endpoint	Description
`GET /user/1`	Returns user `{ id: 1, name: "John Doe" }`
`GET /user/2`	Returns error `{ status: "error", message: "User not found!" }`
`GET /async-error`	Returns error `{ status: "error", message: "Async error occurred!" }`

Key Benefits of This Structure

Strong Type Safety → Avoid runtime errors with proper typings.
Separation of Concerns → Controller, Service, Middleware, and Utility layers.
Centralized Error Handling → Keeps code clean and reusable.
Handles Both Sync & Async Errors → Ensures robustness.
Easy Scalability → New features can be added without modifying core logic.