Backend 2.8_CW Exercises

best practices in express

ex01: global error handler middleware

In this exercise, you'll implement a global error handler middleware to handle errors that might occur during the API request-response cycle.

challenge

1. Create a global error handler middleware.
2. Use this middleware at the end of your middleware chain.
3. Handle any unhandled errors and return an appropriate response.
4. Create Error Handler Middleware: Define a middleware function that takes four parameters: `err`, `req`, `res`, and `next`.
5. Log the Error: Log the error to the console or a logging service for tracking.
6. Send an Error Response: Respond to the client with a suitable error message and status code.

solution

https://replit.com/@tanaypratap/BE28CW-ex01

```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).json({ error: 'Something went wrong' })
})
```

COPY

ex02: versioning the APIs

In this exercise, you'll learn how to implement API versioning to maintain backward compatibility and introduce new features without breaking existing clients.

challenge

1. Create separate route files for different API versions.
2. Use the appropriate route file based on the requested version.
3. Create Versioned Route Files: Create separate route files named with version numbers, e.g., `v1.routes.js`, `v2.routes.js`.
4. Implement Versioned Middleware: In your main `app.js` or entry point, use middleware to direct incoming requests to the appropriate route file based on the requested version.

solution

https://replit.com/@tanaypratap/BE28CW-ex02

```
// v1.routes.js
const express = require('express')
const router = express.Router()

// Define v1 routes

module.exports = router
// app.js
const v1Routes = require('./v1.routes')
app.use('/api/v1', v1Routes)
```

COPY

ex03: global 404 middleware

In this exercise, you'll create a global 404 middleware to handle requests for routes that don't exist.

challenge

1. Create 404 Middleware: Define a middleware function that takes `req` and `res` parameters.
2. Send 404 Response: Respond with a 404 status code and a message indicating that the requested route is not found.

solution

https://replit.com/@tanaypratap/BE28CW-ex03

```
app.use((req, res) => {
  res.status(404).json({ error: 'Route not found' })
})
```

COPY

ex04: setup for CORS

In this exercise, you'll configure Cross-Origin Resource Sharing (CORS) to allow requests from different domains.

challenge

1. Install CORS Middleware: Install the `cors` middleware using `npm install cors`.
2. Use CORS Middleware: In your `app.js`, require and use the `cors` middleware. Configure it to allow specific origins and HTTP methods.

solution

https://replit.com/@tanaypratap/BE28CW-ex04

```
const cors = require('cors')
const allowedOrigins = ['<http://localhost:3000>', '<https://example.com>']

app.use(
  cors({
    origin: (origin, callback) => {
      if (allowedOrigins.includes(origin) || !origin) {
        callback(null, true)
      } else {
        callback(new Error('Not allowed by CORS'))
      }
    },
  }),
)
```

COPY

ex05: Helmet

In this exercise, you'll enhance the security of your Express application by using the `helmet` middleware.

challenge

1. Install Helmet Middleware: Install the `helmet` middleware using `npm install helmet`.
2. Use Helmet Middleware: In your `app.js`, require and use the `helmet` middleware. It will automatically set various security headers to help protect your app.

solution

https://replit.com/@tanaypratap/BE28CW-ex05

```
const helmet = require('helmet')
```

```
app.use(helmet())
```