Backend 2.5_CW Exercises

routers and middlewares

ex01: routers

ex01.1: introducing routers and attaching routes

In this exercise, we'll introduce routers and learn how to attach routes to them.

challenge

1. Creating a Router: In your main `index.js` file, create a router by importing `express.Router()`.

```
const express = require('express')
const carRouter = express.Router()
```

COPY

2. Attaching Routes: Add a GET route to the router that returns JSON data of all cars.

```
carRouter.get('/', (req, res) => {
  res.json(cars);
});

carRouter.get('/:id', () = {})

If I want to update a particular car with it's id through POST request, how will I do it?

carRouter.post(":/id", () => {}) // 23
```

COPY

3. Mounting the Router: In your `index.js` file, mount the router to a specific path (e.g., `/cars`) using `app.use()`.

```
app.use('/cars', carRouter)
```

COPY

solution

https://replit.com/@tanaypratap/BE25CW-ex011

how to test refactor?

YOUR CODE REFACTOR SHOULD NOT CHANGE THE API.

ex01.2: moving router to a separate file

In this exercise, we'll move the router to a separate file and demonstrate the concept of modularization.

challenge

1. Create Router File: Create a new file named `cars.router.js`.
2. Move Router Logic: Move the router logic from `index.js` to `cars.router.js`.
3. Export the Router: In `cars.router.js`, export the router using `module.exports`.
4. Import and Use Router: In `index.js`, import the router using `require()` and use it with `app.use()`.

solution

https://replit.com/@tanaypratap/BE25CW-ex012

ex02: middlewares

ex02.1: introducing and writing simple middlewares

In this exercise, we'll introduce the concept of middleware and demonstrate how to create a simple middleware function.

challenge

Creating a Middleware: Middleware functions are functions that have access to the request (`req`), response (`res`), and next middleware function in the application's request-response cycle.

Let's create a middleware function that logs a message to the console for each incoming request.

solution

https://replit.com/@tanaypratap/BE25CW-ex021

```
// Middleware function
const loggerMiddleware = (req, res, next) => {
  console.log('Incoming request at:', new Date().toISOString())
  next() // Call next to move to the next middleware or route handler
}

// Applying middleware to all routes
app.use(loggerMiddleware)
```

COPY

By using `app.use()` with `loggerMiddleware`, the middleware is applied to all routes in the application.

ex02.2: writing custom middleware for validation

Now, let's write a custom middleware that checks whether a certain condition is met.

challenge

1. Creating Custom Middleware: Define a custom middleware function that checks if the request has a query parameter named `validate`.

solution

https://replit.com/@tanaypratap/BE25CW-ex022

```
// Custom middleware for validation
const validationMiddleware = (req, res, next) => {
  const validateParam = req.query.validate

  if (validateParam === 'true') {
    next() // Proceed to the next middleware or route handler
  } else {
    res.status(403).json({ error: 'Validation failed' }) // Return error response
  }
}

// Applying custom middleware to a specific route
app.get('/cars/:id', validationMiddleware, (req, res) => {
  // Route logic to retrieve car by ID
})
```

COPY

understanding

Here, the `validationMiddleware` is applied only to the route that retrieves a specific car by ID. If the query parameter `validate` is set to `true`, the middleware allows the request to proceed; otherwise, it returns a forbidden error.

ex02.3: writing middleware for a specific route

In this exercise, we'll write middleware that's specific to a particular route.

challenge

1. Creating Route-Specific Middleware: Define a middleware function that checks if the user is authenticated.

solution

https://replit.com/@tanaypratap/BE25CW-ex023

```
function checkAuthentication(req) {
  return req.headers.authorization === 'Bearer validAuthToken'
}
// Route-specific middleware for authentication
const authenticateMiddleware = (req, res, next) => {
  const isAuthenticated = checkAuthentication(req) // Implement your authentication logic

  if (isAuthenticated) {
    next() // Proceed to the next middleware or route handler
  } else {
    res.status(401).json({ error: 'Unauthorized' }) // Return unauthorized error
  }
}

// Applying middleware to a specific route
app.post('/secure-action', authenticateMiddleware, (req, res) => {
  res.json({ message: 'Secure action executed successfully' })
})
```

COPY

In this example, `authenticateMiddleware` is applied only to the route that performs a secure action. If the user is authenticated, the middleware allows the request to proceed; otherwise, it returns an unauthorized error.

ex02.4: writing middleware for an entire router

Lastly, let's explore writing middleware that applies to an entire router.

challenge

1. Creating Router-Level Middleware: Define a middleware function that sets a response header for all routes on the car router.

solution

https://replit.com/@tanaypratap/BE25CW-ex024

```
// Router-level middleware for setting a response header
const responseHeaderMiddleware = (req, res, next) => {
  res.setHeader('X-App-Version', '1.0') // Set a custom response header
  next() // Proceed to the next middleware or route handler
}

// Applying router-level middleware to the entire car router
carRouter.use(responseHeaderMiddleware)
```

COPY

By using `carRouter.use()`, the middleware is applied to all routes mounted on the `carRouter`.

ex02.5: middleware order and execution

In this final exercise, we'll explore how the order of middleware placement affects execution.

challenge

Creating Middleware for Every Route: Write a middleware that logs a message for every incoming request, and apply it using `app.use()`.

solution

https://replit.com/@tanaypratap/BE25CW-ex025

```
// Middleware for logging every incoming request
const logAllRequestsMiddleware = (req, res, next) => {
  console.log('Request received:', req.method, req.url)
  next() // Proceed to the next middleware or route handler
}

// Applying middleware for every route
app.use(logAllRequestsMiddleware)
```

COPY

Placing Middleware: Observe how the placement of this middleware affects the execution order compared to other middleware.