

W3 PRACTICE

Express Basics + POST + Middleware

 *At the end of this practice, you can*

- ✓ **Create** and run a express.js HTTP server
- ✓ **Implement** route handling using express.js
- ✓ Parse form data from POST requests with middleware.
- ✓ Apply middleware concept to logging

 *Get ready before this practice!*

- ✓ **Read** the following documents to understand the nature of Express.js:
<https://expressjs.com/>
- ✓ **Read** the following documents to know more about Express.js's built-in middleware's:
<https://expressjs.com/en/resources/middleware.html>
- ✓ **Read** the following documents to understand MDN: HTTP POST:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods/POST>
- ✓ **Read** the following documents to array filter:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

 *How to submit this practice?*

- ✓ Once finished, push your **code to GITHUB**
- ✓ Join the **URL of your GITHUB** repository on LMS



Student's name: Someth Phay

IDTB100019

SE G2 Gen10 Year2 Term3

Lect. Kim Ang Kheang

S2 - PRACTICE - Native HTTP 1

GitHub Repo:

https://github.com/PhaySometh/Y2_Term3_W3_S2-PRACTICE-ExpressJS_1.git

EXERCISE 1 – Refactoring

Goals

- ✓ Take advantage of Express.js framework's flexibility and minimalism
- ✓ Refactor code from node.js's built-in HTTP Module



Refactor the source code of EXERCISE 2 & 3 in Week 2 to Express.js

Q1 – What challenges did you face when using the native http module that Express.js helped you solve?

➔ **Answer:**

- **Manual routing:** You must check URL and method manually.
- **No body parsing:** You handle request data as raw streams.
- **No middleware:** Must manually implement things like logging or auth.
- **Verbose responses:** Setting headers and sending JSON is tedious.

Q2 – How does Express simplify route handling compared to the native HTTP server?

➔ With native http, routing is done with if checks on req.url and req.method.

With Express, you just use:

```
app.get('/route', handler);
```

It's cleaner, readable, and easier to manage.

Q3 – What does middleware mean in Express, and how would you replicate similar behavior using the native module?

➔ **Express middleware:** Functions run before route handlers using app.use().

Native alternative: You chain functions manually and call next() yourself.

```
function logger(req, res, next) {  
  console.log(req.url);  
  next();  
}
```

Express makes middleware easy and standardized.

EXERCISE 2 – API for Course Records

🔑 For this exercise you will start with a **START CODE (EX-2)**

Goals

- ✓ Understand Route Parameters (:param)
- ✓ Work with Query Parameters (?key=value)
- ✓ Implement Conditional Logic for Filtering
- ✓ Build Real-World Web API Behavior
- ✓ Practice Defensive Programming

Context

You are building a backend API for a university's course catalog. Each course has the following fields

```
{
  "id": "CSE101",
  "title": "Introduction to Computer Science",
  "department": "CS",
  "level": "undergraduate",
  "credits": 3,
  "instructor": "Dr. KimAng",
  "semester": "fall"
}
```

Q1 - Create a route

```
GET /departments/:dept/courses
```

EXAMPLE

```
/departments/CSE/courses
```

Q2 - Accept query parameters to filter the result:

- `level` → e.g., undergraduate, graduate
- `minCredits` → integer
- `maxCredits` → integer
- `semester` → fall, spring, etc.
- `instructor` → partial match

EXAMPLE

```
/departments/CSE/courses?level=undergraduate&minCredits=2&semester=fall
```

Q3 - Return a JSON array of courses that match:

- The `:dept` from the route parameter
- The filter criteria from query parameters

Q4 – Handle Edge Cases

- **Invalid credit ranges** (minCredits > maxCredits)
- **No matching courses**
- **Missing or unsupported** query parameters (ignore them silently)

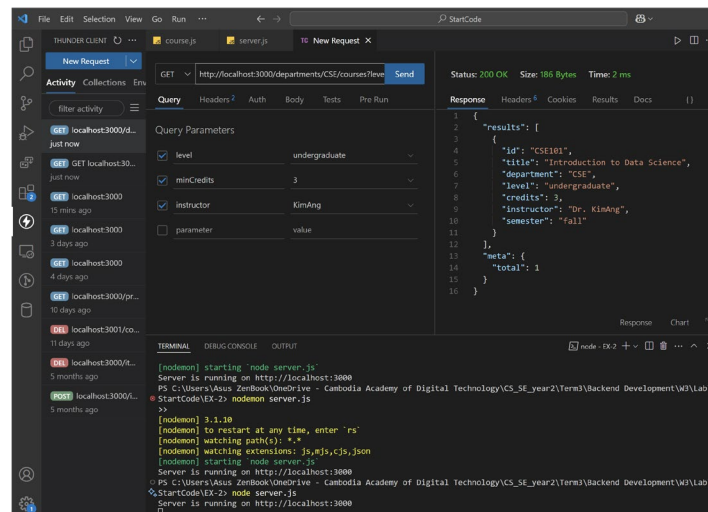
EXAMPLES

REQUEST
/departments/CSE/courses?level=undergraduate&minCredits=3&instructor=KimAng
RESPONSE
<pre>{ "results": [{ "id": "CSE101", "title": "Introduction to Data Science", "department": "CSE", "level": "undergraduate", "credits": 3, "instructor": "Dr. KimAng", "semester": "fall" }], "meta": { "total": 1 } }</pre>

EDGE CASES

- `http://localhost:3000/departments/CSE/courses`
- `http://localhost:3000/departments/CSE/courses?level=undergraduate`
- `http://localhost:3000/departments/CSE/courses?minCredits=4`
- `http://localhost:3000/departments/CSE/courses?instructor=smith&semester=fall`

My response:



EXERCISE 3 – Enhance an API with Middleware

Goal

Your goal is to modularize and secure your course filtering API using **Express middleware**. Middleware helps keep your code clean, reusable, and extensible.

Q1 - Create a middleware function that logs the following for every request:

- HTTP method (GET, POST, etc.)
 - Request path (e.g., /departments/CSE/courses)
 - Query parameters
 - Timestamp in ISO format
- ✓ **Apply this middleware globally** so it logs **all incoming requests** to the server.

Q2 - Create a route-specific middleware to **validate query parameters**:

- If minCredits or maxCredits are present, ensure they are valid integers.
 - If minCredits > maxCredits, return 400 Bad Request with an error message.
- ✓ **Apply this middleware only** to the /departments/:dept/courses route.

Q3 – (Bonus) Token-Based Authentication Middleware

Simulate basic API security:

- Require a token query parameter (e.g., ?token=xyz123)
 - If the token is missing or incorrect, respond with 401 Unauthorized.
- ✓ This middleware can be applied **either globally or to specific routes**.

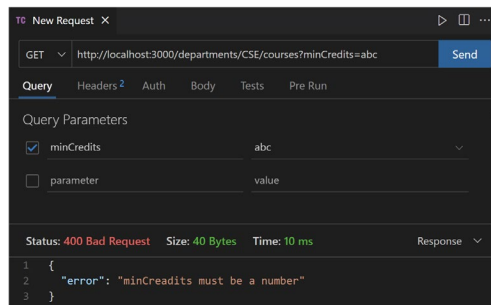
Deliverables

- `logger.js` – contains your logging middleware.
- `validateQuery.js` – contains your validation middleware.
- `auth.js` (optional) – contains your token authentication middleware.
- `server.js` – where you apply middleware and define the course filtering route.

Test cases

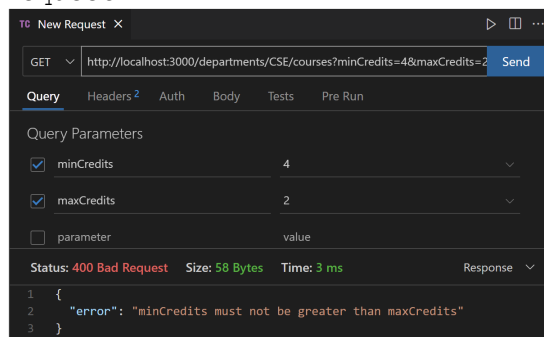
```
GET /departments/CSE/courses?minCredits=abc
```

→ should return 400 Bad Request



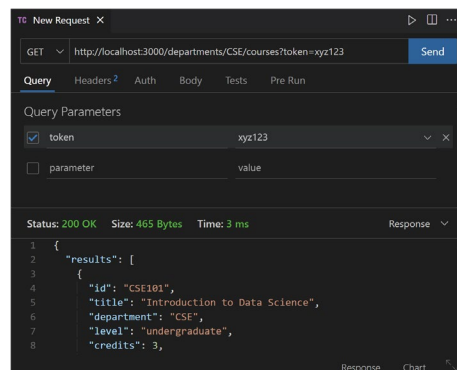
```
GET /departments/CSE/courses?minCredits=4&maxCredits=2
```

→ should return 400 Bad Request



```
GET /departments/CSE/courses?token=xyz123
```

→ should succeed if token middleware is active



My response:

Visual Studio Code interface showing a web application project structure and a REST client request.

EXPLORER

- coursejs
- serverjs EX-2
- loggerjs
- serverjs EX-3
- valid

STARTCODE

- EX-1
- EX-2
 - coursejs
 - package-lock.json
 - package.json
 - serverjs
- EX-3
 - authjs
 - loggerjs
 - serverjs
- validateQueryjs
- node_modules
- package-lock.json
- package.json

serverjs EX-3

```
16 app.get('/departments/:dept/courses', auth, validateQuery, (req, res) => {
32   filtered = filtered.filter(course => course.semester === req.params.semester);
33   if (semester) {
34     filtered = filtered.filter(course => course.semester === semester);
35   }
36   if (instructor) {
37     const keywords = instructor.toLowerCase();
38     filtered = filtered.filter(course =>
39       course.instructor.toLowerCase().includes(keywords));
40   }
41   // Return result
42   res.json({
43     results: filtered,
44     meta: { total: filtered.length }
45   });
46 });
47
48 app.listen(PORT, () => {
49   console.log(`Server is running on http://localhost:${PORT}`);
50 });
```

Terminal

```
url: 'file:///C:/Users/Asus%20ZenBook/OneDrive%20-%20Cambodia%20Academy%20of%20Digital%20Technology/CS_SE_year2/Term3/Backend%20Development/W3/Lab/StartCode/course.js'
Node.js v20.15.0
PS C:\Users\Asus ZenBook\OneDrive - Cambodia Academy of Digital Technology\CS_SE_year2\Term3\Backend Development\W3\Lab\StartCode\EX-3> node server.js
Server is running on http://localhost:3000
[2025-05-16T08:02:18.150Z] GET /departments/CSE/courses Query: [Object: null prototype] { level: 'undergraduate', token: 'xyz123' }
[2025-05-16T08:03:15.681Z] GET /departments/CSE/courses Query: [Object: null prototype] {
  minCredits: '5',
  maxCredits: '2',
  token: 'xyz123'
}
[2025-05-16T08:03:26.684Z] GET /departments/CSE/courses Query: [Object: null prototype] { minCredits: 'abc', token: 'xyz123' }
[2025-05-16T08:03:36.697Z] GET /departments/CSE/courses Query: [Object: null prototype] { level: 'graduate' }
```

New Request

GET http://localhost:3000/departments/CSE/courses?level=graduate

Query Parameters

- level: graduate
- parameter: value

Status: 401 Unauthorized Size: 50 Bytes Time: 2 ms

Response

```
{
  "error": "Unauthorized: Invalid or missing token"
}
```