Day 19 – Express.js Fundamentals, Routing & Middleware

Challenge 1: Setting Up Express Server

User Story:
As a backend developer, I want to set up an Express.js server so that I can handle HTTP requests easily without writing boilerplate code.

Problem Statement:

1. Install Express using npm install express.
2. Create a basic server (server.js) that listens on port **4000**.
3. Add a route / that returns "Welcome to Express Server".

Expected Outcome:
Visiting http://localhost:4000 should display:

```
Welcome to Express Server
```

Bonus:
Add a /status route that returns a JSON object:

```
{ "server": "running", "uptime": "OK" }
```

Self-Evaluation Metrics:

Metric	Target
Installed and imported Express successfully	
Server runs without error	
Properly handled GET request	

Bonus route returning JSON works	

Challenge 2: Express Routing & Query Parameters

User Story:
As a user, I want to search for products by name using query parameters so that I can filter the results dynamically.

Problem Statement:

1. Create a route /products that accepts a query parameter ?name=.
2. If name is provided, respond with "Searching for product: <name>".
3. Otherwise, respond with "Please provide a product name".

Expected Outcome:

- /products?name=Laptop → "Searching for product: Laptop"
- /products → "Please provide a product name"

Bonus:
Return results in JSON format { "query": "<name>" } instead of plain text.

Self-Evaluation Metrics:

Metric	Target
Correctly used req.query	
Implemented conditional logic	
Bonus JSON format implemented	

Challenge 3: Express Middleware

User Story:
As a developer, I want to log every request made to the server so that I can monitor API usage and detect errors.

Problem Statement:

1. Create a custom middleware that logs the HTTP method and URL for each request.
2. Add it globally using app.use().
3. Test by visiting multiple routes.

Expected Outcome:
Console output should show something like:

```
[GET] /products

[GET] /status
```

Bonus:
Add a timestamp for each request in the log.

Self-Evaluation Metrics:

Metric	Target
Middleware correctly logs all routes	
Middleware order applied correctly	
Timestamp formatting correct (Bonus)	

Challenge 4: REST API (CRUD Operations)

User Story:
As an admin, I want to manage a list of books using REST APIs (GET, POST, PUT, DELETE).

Problem Statement:

1. Create routes for /books:

   - GET /books → Returns all books
   - POST /books → Adds a new book
   - PUT /books/:id → Updates book details
   - DELETE /books/:id → Deletes a book

Use an in-memory array to store books like:

```
 let books = [
 { id: 1, title: "1984", author: "Orwell" },
 { id: 2, title: "The Alchemist", author: "Coelho" }
];
```

2.

Expected Outcome:
API responds correctly for all CRUD operations using Postman or curl.

Bonus:
Validate data before adding a book (e.g., ensure title and author exist).

Self-Evaluation Metrics:

Metric	Target
Implemented all CRUD routes	
Used route parameters (req.params)	
Validation works properly	

Challenge 5: Modular Routing & Error Handling

User Story:
As a developer, I want to separate my routes and handle errors gracefully to keep the project modular and maintainable.

Problem Statement:

1. Move all /books routes into a separate file routes/books.js.
2. Import it into server.js using app.use('/books', bookRouter).
3. Add a global error-handling middleware to catch 404 and internal errors.

Expected Outcome:

- Routes are modularized and imported cleanly.
- Invalid routes return a custom error message: "Route not found".

Bonus:
Add a centralized error handler that logs errors and returns { "error": "Internal Server Error" }.

Self-Evaluation Metrics:

Metric	Target
Routes successfully modularized	
Global error handler implemented	
404 handling works as expected	

Extended Practice (Optional 90-min Project)

Mini Project – "BookStore API"

- Combine all challenges into one project.
- Add CORS, validation using express-validator, and a GET /books/:id route.
- Test using Postman collection.

Outcome:
A fully functional mini REST API demonstrating complete Express fundamentals.

Self-Evaluation Summary:

Area
Understanding Express setup
Middleware & routing structure
Error handling & validation
Code modularity
