

# Laporan Praktikum WSE #6

Mata Kuliah : Web Service Engineering

Dosen Pengampu : Muhayat, M.IT

Praktikum : P6 – RESTful API Best Practices (Express)

Nama Mahasiswa : Ruqayah

NIM : 230104040208

Kelas : TI23A

Tanggal Praktikum : 04-11-2025

---

## A. Tujuan Praktikum

1. Memahami penerapan prinsip RESTful pada API Express.
2. Menggunakan HTTP Method dan Status Code secara tepat.
3. Mengimplementasikan 7 RESTful Principles dalam desain endpoint.
4. Menangani validasi input dan error secara terstandar.
5. Menyiapkan dokumentasi endpoint yang mudah dibaca dan diuji.

## B. Lingkungan & Tools

- ✓ Node.js 18+ & npm
- ✓ Express.js
- ✓ VS Code / Postman / Thunder Client
- ✓ Nodemon (dev dependency)
- ✓ morgan → logging request
- ✓ Middleware: `validateProduct.js`, `errorHandler.js`

## C. Arsitektur Singkat

- ✓ Client (Postman / Thunder Client / Frontend nanti) → mengirim HTTP request untuk CRUD + uji validasi + uji error.
- ✓ API Server (Express) → jadi pusat kontrol, menerima request, mem-parsing JSON, meneruskan ke router sesuai path /api/....
- ✓ Router (src/routes/products.routes.js) → mendefinisikan endpoint RESTful lengkap: GET, POST, PUT, PATCH, DELETE, plus /api/health.
- ✓ Controller / Handler → di dalam route, berisi logika ambil data, update, hapus, lalu membentuk response dalam format standar ({ success, message, data }).
- ✓ Middleware Validasi (src/middlewares/validateProduct.js) → dipasang khusus di POST dan PUT untuk menolak request yang tidak punya name atau price → balas 400 Bad Request.
- ✓ Middleware Error Global (src/middlewares/errorHandler.js) → menangkap error tak terduga (typo variabel, throw manual) dan mengembalikan 500 Server error tanpa menjatuhkan server.
- ✓ Data Layer (src/data/products.data.js) → masih pakai array in-memory sebagai sumber data sementara, tapi sudah dipisah agar nanti gampang diganti DB.
- ✓ Utility Response (src/utils/apiResponse.js) → (opsional) menyamakan bentuk response sukses dan error supaya API konsisten.

- ✓ Logging (morgan) → mencatat setiap request (method, path, status) untuk kebutuhan observability di praktikum berikutnya.
- ✓ Response JSON → semua endpoint mengembalikan JSON terstandar + status code yang sesuai (200, 201, 400, 404, 500).

## D. Langkah Implementasi (ringkas)

1. Membuat struktur project Express di folder src/
2. Membuat file products.routes.js dengan endpoint CRUD + PATCH
3. Menambahkan middleware validateProduct untuk validasi POST dan PUT
4. Menambahkan middleware errorHandler untuk menangani error global
5. Menguji endpoint di Postman: GET, POST, PUT, PATCH, DELETE
6. Menguji validasi input dan simulasi error 500
7. Menulis dokumentasi endpoint di README.md

## E. Hasil & Bukti

Lampirkan Screenshot Hasil Uji Endpoint di Postman

- Implementasi CRUD
- GET all --->/api/products (Menampilkan seluruh produk) → 200 OK

```

1 {
2   "success": true,
3   "data": [
4     {
5       "id": 1,
6       "name": "Laptop",
7       "price": "8000000"
8     },
9     {
10       "id": 2,
11       "name": "Headset",
12       "price": "2500000"
13     }
14   ]
15 }
  
```

- GET by id ---> /api/products/1 (Menampilkan produk dengan ID tertentu → 200 OK / 404 Not Found)

Just's Workspace

P6-RESTful-BestPractice-230104040208 / by id

GET http://localhost:3000/api/products/1

Body: { } JSON

```

1 {
2   "success": true,
3   "data": {
4     "id": 1,
5     "name": "Laptop",
6     "price": "8800000"
7   }
8 }
```

200 OK | 12 ms | 301 B | Save Response

POST new product --->/api/products (Berhasil tambah produk) → 201 Created

Just's Workspace

P6-RESTful-BestPractice-230104040208 / new products

POST http://localhost:3000/api/products

Body: { } JSON

```

1 {
2   "success": true,
3   "message": "Product created",
4   "data": {
5     "id": 1762157313515,
6     "name": "keyboard RGB",
7     "price": 250000
8 }
```

201 Created | 38 ms | 350 B | Save Response

PUT id 1 --->/api/products/1 (Berhasil update produk) → 200 OK

Just's Workspace

P6-RESTful-BestPractice-230104040208 / id 1

PUT http://localhost:3000/api/products/1

Body: { } JSON

```

1 {
2   "success": true,
3   "message": "Product updated",
4   "data": {
5     "id": 1,
6     "name": "laptop pro X",
7     "price": 12000000
8 }
```

200 OK | 6 ms | 334 B | Save Response

## PATCH id 1 --->/api/products/1(Ubah sebagian data) → 200 OK

The screenshot shows the Postman interface with a collection named "P6-RESTful-BestPractice-230104040208". A PATCH request is made to `http://localhost:3000/api/products/1`. The body contains the following JSON:

```
1 {  
2   "price": 9900000  
3 }
```

The response status is 200 OK, with a response time of 5 ms and a response size of 344 B. The response body is:

```
1 {  
2   "success": true,  
3   "message": "Product partially updated",  
4   "data": {  
5     "id": 1,  
6     "name": "Laptop pro X",  
7     "price": 9900000  
8   }  
9 }
```

## DELETE id 1 --->/api/products/1(Produk terhapus) → 200 OK

The screenshot shows the Postman interface with a collection named "P6-RESTful-BestPractice-230104040208". A DELETE request is made to `http://localhost:3000/api/products/1`. The headers section includes a Content-Type header set to application/json.

The response status is 200 OK, with a response time of 5 ms and a response size of 279 B. The response body is:

```
1 {  
2   "success": true,  
3   "message": "Product deleted"  
4 }
```

- Standarisasi Response & Validasi

## POST No Name --->/api/products

The screenshot shows the Postman interface with a POST request to `http://localhost:3000/api/products`. The request body is raw JSON with fields `name`, `price`, and `stock`. The response is a 400 Bad Request with validation errors:

```
{ "success": false, "message": "Validation error", "errors": [ { "field": "name", "message": "Name is required" } ] }
```

## POST No Price --->/api/products

The screenshot shows the Postman interface with a POST request to `http://localhost:3000/api/products`. The request body is raw JSON with fields `name` and `stock`. The response is a 400 Bad Request with validation errors:

```
{ "success": false, "message": "Validation error", "errors": [ { "field": "price", "message": "Price is required" } ] }
```

## PUT No Name --->/api/products/1

POST /products

```

1 {
2   "stock": 5
3 }

```

Body: {"success": false, "message": "Validation error", "errors": [{"field": "name", "message": "Name is required"}, {"field": "price", "message": "Price is required"}]}

- Error Handling & Logging

GET /products/crash/test

Body: {"success": false, "message": "Server error"}

- Endpoint Health Check

localhost:3000/api/health

```
"status": "ok", "time": "2025-11-03T15:36:44.130Z"}
```

## **F. Analisis**

API telah memenuhi 7 RESTful Principles, memiliki struktur modular, dan menampilkan status code yang konsisten. Middleware validasi berhasil mencegah input kosong, dan error handler menampilkan pesan 500 tanpa mematikan server.

## **G. Kesimpulan**

Melalui praktikum ini, dapat dipahami bahwa RESTful API tidak hanya tentang CRUD, tetapi juga tentang penerapan prinsip desain yang konsisten, aman, dan mudah digunakan oleh client.

## **H. Checklist Praktikum**

- ✓ Endpoint CRUD lengkap
- ✓ PATCH berfungsi
- ✓ Middleware validasi aktif
- ✓ Error handler berjalan
- ✓ Status code konsisten
- ✓ Dokumentasi README.md selesai