Coding & Development / JavaScript

Generate Express.js API endpoints with routing, middleware, request validation, and response formatting.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: API Development, Backend, Express.js
Updated: May 2026
Why This Prompt Exists
Most API endpoints are written from scratch — repetitive boilerplate for every route.

You get:

- no request validation (bad data crashes)
- inconsistent response formats (hard to document)
- no error handling (500 errors for bad input)
- no authentication (security holes)
- no rate limiting (abuse vectors)

But an API endpoint is not just a function.

It is a contract with validation, error handling, and security.

- Request validation: check params, query, body
- Authentication: verify user/permissions
- Business logic: the actual work
- Response formatting: consistent structure
- Error handling: catch and respond with correct status codes

Without structure, APIs are buggy and inconsistent.

This framework forces AI to generate production-ready endpoints.

The Prompt

```
Assume the role of a backend engineer who builds robust, secure API endpoints.

Your task is to generate an Express.js endpoint.

Generate:

1. ROUTE SETUP
   - HTTP method (GET, POST, PUT, DELETE)
   - Path with parameters

2. REQUEST VALIDATION
   - Validate URL parameters
   - Validate query string
   - Validate request body (using Joi, zod, or custom)

3. AUTHENTICATION (if needed)
   - Verify API key or JWT
   - Check user permissions

4. BUSINESS LOGIC
   - Database query or external API call
   - Data transformation

5. RESPONSE FORMATTING
```

```
   - Success response (200/201)
   - Error responses (400, 401, 403, 404, 500)

6. ERROR HANDLING
   - Try/catch wrapper
   - Specific error messages

INPUTS:

HTTP Method:
[GET / POST / PUT / DELETE / PATCH]

Endpoint Path:
[E.G., "/users/:id"]

Request Body Schema (if POST/PUT):
[DESCRIBE FIELDS AND TYPES]

Authentication Required:
[YES / NO]

Database Operation:
[E.G., "Find user by ID from PostgreSQL"]

Response Format:
[E.G., "Return user object with id, name, email"]

RULES:
- Validate all inputs before processing
```

```
- Use HTTP status codes correctly (200, 201, 400, 401, 403, 404, 500)
- Return consistent error format (e.g., { error: { message, code } })
- Don't leak internal details in error messages
- Use async/await for database/API calls
- Add rate limiting for public endpoints
- Log errors for debugging (but don't expose to client)
```

How To Use It

- Be specific about the request body schema (field names, types, required).
- Describe the database operation — what data to fetch or update.
- Add authentication details if needed (JWT, API key, session).
- Test the endpoint with valid and invalid inputs.
- Add OpenAPI/Swagger documentation after generation.

Example Input

HTTP Method: POST

Endpoint Path: /api/users

Request Body Schema: name (string, required), email (string, required, valid email), password (string, required, min 8 chars)

Authentication Required: NO (public registration)

Database Operation: Insert new user into PostgreSQL, hash password with bcrypt

Response Format: Return created user object with id, name, email (excluding password)

Why It Works
Most API endpoints are missing validation and error handling.

This framework improves outcomes by forcing:

- request validation (security)
- authentication checks (authorization)
- consistent response formatting (API contract)
- error handling (robustness)
- HTTP status codes (correctness)

Great API endpoints don’t just work — they validate, authenticate, and handle errors consistently.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, JavaScript frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The DOM Manipulation Script