Coding & Development / API Development

Create standardized error response formatting and handling for API endpoints (400, 401, 403, 404, 500).
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Error Handling, API Responses, Backend Development
Updated: May 2026
Why This Prompt Exists
Most APIs have inconsistent error responses — making them hard to consume.

You get:

- string errors (“User not found”) — no structure
- HTML error pages for JSON APIs (unparseable)
- no error codes (can’t programmatically handle)
- stack traces exposed to clients (security risk)
- different error formats across endpoints

But error handling is not afterthought.

It is part of your API contract.

- Consistent format: { error: { code, message, details } }
- HTTP status codes: 400, 401, 403, 404, 422, 500
- Error codes: unique identifiers for error types
- Validation errors: field-specific messages
- Development vs. production: stack traces only in dev

Without standardized errors, API clients are fragile.

This framework forces AI to build consistent error handlers.

The Prompt

```
Assume the role of an API developer who builds consistent error
handling.

Your task is to create an API error handler.

Generate:

1. ERROR RESPONSE FORMAT
   - Consistent structure for all errors
   - Example: { error: { code, message, details, timestamp } }

2. ERROR CODES
   - Unique identifiers for each error type
   - e.g., "USER_NOT_FOUND", "INVALID_INPUT", "UNAUTHORIZED"

3. HTTP STATUS CODE MAPPING
   - Which error codes map to which HTTP statuses

4. ERROR HANDLER MIDDLEWARE
   - Catches all errors
   - Formats response
   - Logs error (without exposing to client)

5. CUSTOM ERROR CLASSES
   - Extend built-in Error
```

```
   - Add code, statusCode, details

6. VALIDATION ERROR HANDLER
   - Field-specific error messages

INPUTS:

Framework:
[EXPRESS / FASTAPI / DJANGO / OTHER]

Error Format Preference:
[ { error: { message, code } } / { success: false, error } / CUSTOM ]

Include Stack Traces in Production:
[YES / NO]

Logging Requirement:
[LOG TO FILE / LOG TO CONSOLE / LOG TO SERVICE]

RULES:
- Never expose stack traces to clients in production
- Use consistent error format across all endpoints
- Include error codes for programmatic handling
- Log errors for debugging (but don't leak internals)
- Return appropriate HTTP status codes
- Handle validation errors with field-specific messages
- Distinguish between client errors (4xx) and server errors (5xx)
```

How To Use It

- Never expose stack traces in production — use environment flag.
- Use consistent error format across all endpoints.
- Include error codes for programmatic error handling.
- Log errors to a file or service for debugging.
- Distinguish between client errors (4xx) and server errors (5xx).

Example Input

Framework: EXPRESS

Error Format Preference: { error: { message, code } }

Include Stack Traces in Production: NO

Logging Requirement: LOG TO CONSOLE (development) + LOG TO FILE (production)

Why It Works
Most APIs have inconsistent error responses.

This framework improves outcomes by forcing:

- consistent format (predictability)
- error codes (programmable handling)
- status code mapping (HTTP semantics)
- error logging (debugging)
- validation errors (user feedback)

Great error handlers don’t just report problems — they help clients understand and fix them.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, API development 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 Authentication Middleware Builder