Coding & Development / API Development

Design a REST API specification (endpoints, methods, request/response schemas) from a described use case.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: API Design, RESTful Architecture, Backend Planning
Updated: May 2026
Why This Prompt Exists
Most APIs are designed ad hoc — endpoints added randomly without a consistent pattern.

You get:

- inconsistent naming (/getUser, /fetchProducts)
- wrong HTTP methods (GET for mutations)
- no status code standards (200 for everything)
- missing endpoints (can’t complete workflows)
- duplicate functionality across endpoints

But API design is not endpoint collection.

It is a consistent interface for data operations.

- Resources: nouns (users, products, orders)
- HTTP methods: GET, POST, PUT, PATCH, DELETE
- Status codes: 200, 201, 400, 401, 403, 404, 500
- Naming: plural nouns, kebab-case, consistent patterns
- Relationships: nested resources (/users/:id/orders)

Without design, your API becomes a mess.

This framework forces AI to design consistent, RESTful APIs.

The Prompt

```
Assume the role of an API architect who designs RESTful, consistent
APIs.

Your task is to design a REST API specification.

Generate:

1. RESOURCE LIST
   - Main resources (nouns) in the system

2. ENDPOINT TABLE
   For each resource:
   - HTTP method
   - Endpoint path
   - Description
   - Request body schema (if applicable)
   - Response schema
   - Status codes

3. RELATIONSHIPS
   - Nested resources (e.g., /users/:id/orders)
   - How to handle related data

4. QUERY PARAMETERS
   - Filtering, sorting, pagination
```

```
   - Field selection

5. ERROR RESPONSES
   - Standard error format
   - Common error codes

6. AUTHENTICATION
   - How to authenticate (API key, JWT, OAuth)

INPUTS:

Use Case Description:
[WHAT PROBLEM DOES THE API SOLVE?]

Resources (main entities):
[LIST]

Operations Needed (CRUD or custom):
[E.G., "Create user, update profile, list orders"]

Authentication Required:
[YES / NO]

Data Volume (for pagination):
[SMALL (<1K) / MEDIUM (1K-100K) / LARGE (100K+)]

RULES:
- Use plural nouns for resource names (/users not /user)
- Use HTTP methods correctly (GET for reads, POST for creates, etc.)
```

```
- Use nested resources for related data (/users/:id/orders)
- Return appropriate status codes (201 for creation, 404 for not
found)
- Use consistent error format (e.g., { error: { code, message } })
- Include pagination for list endpoints (limit, offset or cursor)
- Use query parameters for filtering and sorting
```

How To Use It

- Describe the use case clearly — what will developers build with this API?
- List all resources (nouns) upfront — users, products, orders, etc.
- Specify which operations are needed for each resource.
- Consider pagination early if data volume is high.
- Document the API after design (OpenAPI/Swagger).

Example Input

Use Case Description: A task management API for teams to create projects, assign tasks, track status, and comment on tasks.

Resources: users, projects, tasks, comments

Operations Needed: CRUD for all resources, assign users to tasks, change task status, add comments to tasks

Authentication Required: YES (JWT)

Data Volume: MEDIUM (1K-100K tasks per project)

Why It Works
Most APIs are designed ad hoc.

This framework improves outcomes by forcing:

- resource identification (clarity)
- HTTP method correctness (standards)
- status code usage (predictability)
- relationship handling (nesting)
- pagination planning (scalability)

Great API design doesn't just work — it's consistent, predictable, and RESTful.

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 API Client SDK Generator