Coding & Development / Python Prompts

Create a Python class that wraps an external API with methods for authentication, endpoints, and error handling.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: API Integration, SDK Development, HTTP Clients
Updated: May 2026
Why This Prompt Exists
Most API integration code is repetitive and scattered — not reusable.

You get:

- raw requests calls everywhere (duplication)
- no authentication handling (hardcoded tokens)
- poor error handling (unhandled HTTP errors)
- no rate limiting (get blocked)
- no retry logic (transient failures crash)

But an API wrapper is not a collection of requests.

It is a reusable client with consistent patterns.

- Authentication: API key, OAuth, or token
- Session management: reuse connection, handle headers
- Endpoint methods: named methods for each API call
- Error handling: raise custom exceptions for HTTP errors
- Rate limiting: respect API limits with backoff

Without a wrapper, API integration is fragile.

This framework forces AI to build robust API clients.

The Prompt

```
Assume the role of a Python developer who builds robust API wrappers.

Your task is to create a Python class that wraps an external API.

Generate:

1. CLASS DEFINITION
   - __init__ with authentication
   - Session management

2. AUTHENTICATION METHOD
   - API key, OAuth, or token handling

3. REQUEST METHOD (_request)
   - Handles HTTP methods
   - Adds headers, params
   - Error handling (timeout, status codes)

4. ENDPOINT METHODS (3-5)
   - Named methods for each API endpoint
   - Type hints for parameters and returns
   - Docstrings

5. ERROR HANDLING
   - Custom exception classes
```

   - HTTP status code handling

6. USAGE EXAMPLE
   - How to instantiate and use the wrapper

INPUTS:

API Name:
[INSERT]

Base URL:
[INSERT]

Authentication Type:
[API KEY / BEARER TOKEN / OAUTH / BASIC AUTH]

Endpoints to Wrap (3-5 with descriptions):
[LIST]

Rate Limit (if known):
[E.G., "100 requests per minute"]

RULES:
- Use requests.Session for connection reuse
- Raise custom exceptions for HTTP errors (not just print)
- Include timeout parameter for all requests
- Add retry logic for transient failures (5xx errors)
- Type hints for all public methods
- Docstrings for all public methods

```
- Keep API keys out of code (use environment variables)
```

How To Use It

- Get API documentation before using this prompt (base URL, endpoints, auth).
- Test the generated wrapper with a real API key (use a test environment).
- Add logging for debugging production issues.
- Handle pagination if the API uses it (the prompt doesn't auto-handle).
- Add async support if needed (separate prompt).

Example Input

API Name: GitHub API

Base URL: https://api.github.com

Authentication Type: BEARER TOKEN (Personal Access Token)

Endpoints to Wrap: GET /user (get authenticated user), GET /user/repos (list user repos), GET /repos/{owner}/{repo} (get specific repo), POST /repos/{owner}/{repo}/issues (create issue)

Rate Limit: 5000 requests per hour

Why It Works
Most API integration code is fragile and scattered.

This framework improves outcomes by forcing:

- session management (efficiency)
- authentication handling (security)
- error handling (robustness)
- rate limiting (compliance)

- retry logic (resilience)

Great API wrappers don’t just work — they fail gracefully and handle edge cases.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, Python 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 Python Web Scraper Builder