CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

10 REST API BEST PRACTICES

CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

1. Use descriptive and meaningful resource names

Instead of generic or ambiguous names, choose resource names that accurately represent the entities they represent

Bad

```
/api/users/123
```

● ●

Good

```
/api/customers/123
```

CODEWITH_RAJESH
RAJESH KUMAR

SWIPE »»

2. Use HTTP methods correctly

Use the appropriate HTTP methods (GET, POST, PUT, DELETE, etc.) for different operations

GET	**Retrieve a resource**
POST	Create a new resource
PUT	Update an existing resource
DELETE	Delete a resource

CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

3. Use HTTP status codes correctly

Return the appropriate HTTP status codes to indicate the success or failure of an API request

200	**OK: Successful GET request**
201	Created: Successful resource creation
404	Not Found: Resource not found
400	Bad Request: The request was invalid or cannot be served
401	Unauthorized: The request requires authentication
403	Forbidden: The server understands the request, but it refuses to authorize it.
500	Internal Server Error: Server error

4. Version your APIs

Use versioning to ensure backward compatibility and allow for future enhancements without breaking existing clients.

Versioning Method	Description	Format
URL path	The version number is included in the URL path, such as "api/v1/resources"	api/v1/resources
URL query parameter	The version number is included as a parameter in the URL, such as "api?version=1"	api?version=1
Custom request header	A custom header, such as "X-API-Version" is used to specify the version number	X-API-Version: 1
Content negotiation	The client specifies the desired version through the use of the "Accept" header in the request	Accept: application/vnd.company.resource-v1+json

CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

5. Pick your JSON field naming convention (and stick to it)

JSON standard doesn't impose a field naming convention, but it's a best practice to pick one and stick with it.

Convention	Description	Example
snake_case	Lowercase letters with underscores separating words	"user_name"
camelCase	Lowercase first letter of first word, capitalizing subsequent words	"userName"
PascalCase	Capitalizing the first letter of each word	"UserName"
kebab-case	Lowercase letters with hyphens separating words	"user-name"

6. Use query parameters for filtering, sorting, and searching

Query parameters allow you to provide additional information in the URL of an HTTP request to control the response returned by the server.

Action	Description	Example
Filter	Return only the relevant results for a specific request	/users?name={name}
Sort	Order the results in a specific manner	/users?sort_by=age
Paginate	Divide the results into manageable parts or pages	/users?page={page_number}&per_page={results_per_page}

CODEWITH_RAJESH
RAJESH KUMAR

SWIPE >>>

7. Use consistent error messages

In most cases, HTTP status codes are not enough to explain what went wrong.
To help your API consumers, include a structured JSON error message.
The response should include the following information:

Error code: A machine-readable error code that identifies the specific error condition.
Error message: A human-readable message that provides a detailed explanation of the error.
Error context: Additional information related to the error, such as the request ID, the request parameters that caused the error, or the field(s) in the request that caused the error.
Error links: URLs to resources or documentation that provide additional information about the error and how it can be resolved.
Timestamp: The time when the error occurred.

8. Do not maintain state

A REST API should not maintain a state on the server. That's the responsibility of the client.

This is important because it allows for the API to be cacheable, scalable, and decoupled from the client.

For example, an e-commerce API might use cookies to maintain the state of a shopping cart. However, such an approach violates key the key principle of RESTful APIs - they need to be stateless.

CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

9. Implement authentication and authorization

Secure your APIs by implementing proper authentication and authorization mechanisms.

- USE API KEYS, TOKENS, OR OAUTH 2.0 FOR AUTHENTICATION
- APPLY ROLE-BASED ACCESS CONTROL (RBAC) FOR AUTHORIZATION

CODEWITH_RAJESH

RAJESH KUMAR

SWIPE >>>

10. Document your APIs

Provide comprehensive documentation for your APIs, including endpoint details, request/response examples, and usage guidelines

- **SWAGGER/OPENAPI DOCUMENTATION**
- **MARKDOWN-BASED DOCUMENTATION (E.G., USING TOOLS LIKE SWAGGER UI OR REDOC)**

❤ Thanks for reading!

Stay up-to-date with the latest advancements in Java full stack development by following me on the handles below, where I'll be sharing my expertise and experience in the field.

codewith_rajesh

Rajesh Kumar

Like | Save | Share