



Designing RESTful API Quick Guide

v1.0

A downloadable resource of the
Designing RESTful APIs course.

A guide to design web APIs that
follows REST principles using a step-
by-step approach.

Preface

Hi there!

I hope you are doing good.

I've created this cheat sheet to help you with your everyday programming in API. You can **use this as a reference document** while designing an API for the requirements in hand. It shows you each step that you need to take to design a RESTful API from scratch. Moreover, it includes additional tips on naming convention, recommended HTTP Status Codes to include, etc., that you can use for reference.

I personally refer to the first 8 pages of this guide whenever I design an API at my work.

Designing an API is the first step you need to do when working with APIs. **This downloadable resource is part of the Designing RESTful APIs course**, which covers the essentials of designing concepts that any API programmer **must** know. [Click here](#) to know more about the companion course.

See you in the course video!
Praveen.

Table of Contents

Preface	2
Table of Contents	3
Steps to Design an API from Scratch	5
Getting Started with Designing APIs	5
STEP 1: Create a New API	5
STEP 2: Identify the Type of API	5
Overview of RESTful APIs	5
STEP 3: Identify the Server Base URL	5
Designing API Resources	5
STEP 4: Identify the Resources	5
STEP 5: Have the Resources as Plural	5
STEP 6: Define the Resource Models	6
STEP 7: Select the Identifier for Each Resource	6
Designing Associations between Resources	6
STEP 8: Identify the Association for Each Resource	6
STEP 9: Check for the URL Complexity	7
Designing API Operations	7
STEP 10: Identify the Operations for Each Resource	7
Designing API Requests	8
STEP 11: Identify the Parameters Required for the Operation	8
STEP 12: Identify the Content Type of Request for the Operation	8
STEP 13: Identify the Request Body for the Operation	8
Designing API Responses	9
STEP 14: Identify the HTTP Status Codes for the Operation	9
STEP 15: Identify the Content Type of Response for the Operation	9
STEP 16: Identify the Response Body for the Operation	10
STEP 17: Handle Errors for the Operation	11
Design for Filtering, Pagination, and Sorting	11
STEP 18: Identify the Need for Filtering and Add If Needed	11
STEP 19: Identify the Need for Pagination and Add If Needed	11
STEP 20: Identify the Need for Sorting and Add If Needed	12
Designing API Versions	12
STEP 21: Identify the API Versioning Scheme and Set the API Version	12

camelCase or PascalCase or under_scores or hyphens(-)	13
Error Message Format (Full Model)	14
HTTP Status Codes (Recommended)	16
HTTP Status Codes (Complete List)	17

Steps to Design an API from Scratch

Getting Started with Designing APIs

STEP 1: Create a New API

Title: OpenAPI Specification for CMS

Description: API Specification document of the CMS system

Contact: Praveenkumar Bouna (<http://myorganization.com/staff/praveenkumar-bouna>)

Version: 1.0

STEP 2: Identify the Type of API

public

Overview of RESTful APIs

STEP 3: Identify the Server Base URL

`http://{hostname}:{portnumber}/{directory}`

<http://localhost:44333>

Designing API Resources

STEP 4: Identify the Resources

course

student

STEP 5: Have the Resources as Plural

courses (/api/courses)

students (/api/students)

course-subjects (/api/course-subjects)

colleges (/api/colleges)

STEP 6: Define the Resource Models

Course Model

- Course Id
- Course Name
- Course Duration
- Course Type

Student Model

- Student Id
- First Name
- Last Name
- Phone Number
- Address

STEP 7: Select the Identifier for Each Resource

Course Model

- Course Id (IDENTIFIER)
- Course Name
- Course Duration
- Course Type

Student Model

- Student Id (IDENTIFIER)
- First Name
- Last Name
- Phone Number
- Address

Designing Associations between Resources

STEP 8: Identify the Association for Each Resource

Courses

- /api/courses/{courseId}/students
- /api/courses/{courseId}/course-subjects

Students

- None

STEP 9: Check for the URL Complexity

Should not be more complex than collection/item/collection
Combine related resources if required

Courses

- /api/courses
- /api/courses/{courseId}
- /api/courses/{courseId}/students

Students

- /api/students
- /api/students/{studentId}

Designing API Operations

STEP 10: Identify the Operations for Each Resource

- /api/courses
 - GET
 - POST
- /api/courses/{courseId}
 - GET
 - PUT
 - DELETE
- /api/courses/{courseId}/students
 - GET
 - POST
- /api/students
 - GET
 - POST
- /api/students/{studentId}
 - GET
 - PUT
 - DELETE

Designing API Requests

STEP 11: Identify the Parameters Required for the Operation

Query parameters

None

Path parameters

courseId - Unique Course ID of the course model (applicable for individual item).

Header

None

Cookie

None

STEP 12: Identify the Content Type of Request for the Operation

application/json

STEP 13: Identify the Request Body for the Operation

/api/courses

GET

None

POST

courseName
courseDuration
courseType

/api/courses/{courseId}

GET

None

PUT

courseName
courseDuration
courseType

DELETE

None


```
/api/courses/{courseId}/students
  GET
    None
  POST
    firstName
    lastName
    phoneNumber
    address
```

Designing API Responses

STEP 14: Identify the HTTP Status Codes for the Operation

```
/api/courses
  GET
    HTTP 200 OK
  POST
    HTTP 201 CREATED
    HTTP 400 BAD REQUEST (arts)
```

```
/api/courses/{courseId}
  GET
    HTTP 200 OK
    HTTP 404 NOT FOUND (50)
  PUT
    HTTP 200 OK
    HTTP 404 NOT FOUND
  DELETE
    HTTP 204 NO CONTENT (2)
    HTTP 404 NOT FOUND (50)
```

```
/api/courses/{courseId}/students
  GET
    HTTP 200 OK
  POST
    HTTP 201 CREATED
    HTTP 400 INVALID INPUT
```

STEP 15: Identify the Content Type of Response for the Operation

```
application/json
```

STEP 16: Identify the Response Body for the Operation

/api/courses

GET

(array)

courseId

courseName

courseDuration

courseType

POST

courseId

courseName

courseDuration

courseType

/api/courses/{courseId}

GET

courseId

courseName

courseDuration

courseType

PUT

courseId

courseName

courseDuration

courseType

DELETE

None

/api/courses/{courseId}/students

GET

(array)

studentId

firstName

lastName

phoneNumber

address

POST

studentId

firstName

lastName

phoneNumber

address

STEP 17: Handle Errors for the Operation

HTTP 400 BAD REQUEST

```
{
  "error": {
    "code": "INVALID_INPUT",
    "message": "One or more input arguments are invalid",
    "target": "CollegeInfo",
    "details": [
      {
        "code": "INCORRECT_FORMAT",
        "target": "zipcode",
        "message": "Zipcode doesn't follow correct format",
      }
    ]
    "innererror": {
      "message": "Input string wasn't in a correct format",
    }
  }
}
```

Design for Filtering, Pagination, and Sorting

STEP 18: Identify the Need for Filtering and Add If Needed

GET /api/courses

Request:

courseType:

- Support for Filtering.

STEP 19: Identify the Need for Pagination and Add If Needed

GET /api/courses

Request:

page:

- Support for pagination.

size:

- Support for pagination.

STEP 20: Identify the Need for Sorting and Add If Needed

GET /api/courses

Request:

sortBy:

- Support for sorting.

Designing API Versions

STEP 21: Identify the API Versioning Scheme and Set the API Version

Versioning Scheme Used: URL Versioning

Version: 1.0

camelCase or PascalCase or under_scores or hyphens(-)

The below guide will help you when you are confused about which naming convention to use for your API design and documentation.

Part of HTTP Request	Usage (if required)	Example
Resources	hyphens	/api/ courses /api/ human-resources /api/college/{collegeld}/ calculate-tax
Query parameters	camelCase	/api/courses? sort =courseId /api/courses? sortBy =courseId
Query parameter assignment fields	camelCase	/api/courses?sort={ courseDuration }
	camelCase (CAPS for two letter words)	/api/courses?sort= courseId
Headers	Hyphenated PascalCase	Content-Type =application/json
	Hyphenated PascalCase (CAPS for acronyms)	X-API-Version =1.2
Response Body	camelCase (JSON)	{ "courseId" : 1, "courseName" : "Computer Science", "courseDuration" : 4, "courseType" : "Engineering" }

Error Message Format (Full Model)

Below is the format recommended by Microsoft for their APIs. The mandatory parameters are marked with a **bold** face.

```
{
  "error": {
    "code": "XXX",
    "message": "XXX",
    "target": "XXX",
    "details": [
      {
        "code": "XXX",
        "message": "XXX",
        "target": "XXX"
      }
    ]
    "innerError": {
      "message": "XXX"
    }
  }
}
```

Example 1:

```
{
  "error": {
    "code": "INVALID_INPUT",
    "message": "One or more input arguments are invalid"
  }
}
```

Example 2:

```
{
  "error": {
    "code": "INVALID_INPUT",
```

```
"message": "One or more input arguments are invalid",
"target": "CollegeInfo",
"details": [
  {
    "code": "INCORRECT_FORMAT",
    "target": "zipcode"
    "message": "Zipcode doesn't follow the correct format",
  }
]
"innerError": {
  "message": "Input string wasn't in a correct format",
}
}
}
```

HTTP Status Codes (Recommended)

2XX Successful Status Codes

Status Code	Summary
200	OK
201	Created
204	No Content

4XX Client Error Status Codes

Status Code	Summary
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed

5XX Server Error Status Codes

Status Code	Summary
500	Internal Server Error
501	Not Implemented

HTTP Status Codes (Complete List)

1XX Informational Status Codes

Status Code	Summary
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints

2XX Successful Status Codes

200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

4XX Client Error Status Codes

405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict

3XX Redirection Status Codes

Status Code	Summary
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	(Unused)
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error Status Codes

Status Code	Summary
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found

5XX Server Error Status Codes

Status Code	Summary
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable

[Designing RESTful APIs: Learn to Design API from Scratch](#)

410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Long
415	Unsupported Media Type
416	Range Not Satisfiable
417	Expectation Failed
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
425	Too Early
426	Upgrade Required
427	Unassigned
428	Precondition Required
429	Too Many Requests
430	Unassigned
431	Header Fields Too Large
451	Unavailable For Legal Reasons

504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
509	Unassigned
510	Not Extended
511	Network Authentication Required

Thank you!

I hope this resource was helpful to you.