API DEVELOPMENT PATH

DESIGNING RESTFUL API QUICK GUIDE

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 well.

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 at 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 conventions, 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.

Change Log

Version	Date	Remarks
v1.0	18-Mar-2021	Initial version
v1.1	1-Sep-2021	Added the endpoints for the resource along with that of association resources.
v1.2	22-Jan-2022	Added assignment solutions - Added the endpoints for students' resources.

Table of Contents

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: int
    - Course Name: string
    - Course Duration: int
    - Course Type: string

- Student Model
    - Student Id: int
    - First Name: string
    - Last Name: string
    - Phone Number: string
    - Address: string

STEP 7: Select the Identifier for Each Resource

- Course Model
    - Course Id: int **(IDENTIFIER)**
    - Course Name: string
    - Course Duration: int
    - Course Type: string

- Student Model
    - Student Id: int **(IDENTIFIER)**
    - First Name: string
    - Last Name: string
    - Phone Number: string
    - Address: string

Designing Associations between Resources

STEP 8: Identify the Association for Each Resource

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

- Students
    - /api/students

/api/students/{studentId}

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

Requests:

Query parameters

None

Path parameters

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

studentId - Unique Student ID of the student model (applicable for individual items).

Header

None

Cookie

None

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

Content-Type: application/json

STEP 13: Identify the Request Body for the Operation

```
/api/courses
        GET
                Request Body:
                        None
        POST
                Request Body:
                        courseName
                        courseDuration
                        courseType

/api/courses/{courseId}
        GET
                Request Body:
                        None
        PUT
                Request Body:
                        courseName
                        courseDuration
                courseType
        DELETE
            Request Body:
                None

    /api/courses/{courseId}/students
        GET
            Request Body:
                None
        POST
            Request Body:
                firstName
                lastName
                phoneNumber
                address

    /api/students
        GET
            Request Body:
                None
        POST
            Request Body:
                firstName
                lastName
                phoneNumber
                address

    /api/students/{studentId}
        GET
            Request Body:
                None
        PUT
            Request Body:
                firstName
                lastName
                phoneNumber
                address
        DELETE
            Request Body:
                None
```

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

/api/courses/{courseId}
        GET
                HTTP 200 OK
                HTTP 404 NOT FOUND
        PUT
                HTTP 200 OK
                HTTP 404 NOT FOUND
        DELETE
                HTTP 204 NO CONTENT
                HTTP 404 NOT FOUND

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

/api/students
        GET
                HTTP 200 OK
        POST
                HTTP 201 CREATED
                HTTP 400 BAD REQUEST

/api/students/{studentId}
        GET
                HTTP 200 OK
                HTTP 404 NOT FOUND
        PUT
                HTTP 200 OK
                HTTP 404 NOT FOUND
        DELETE
                HTTP 204 NO CONTENT
                HTTP 404 NOT FOUND
```

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

Content-Type: 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

/api/students
        GET
                (array)
                studentId
                firstName
                lastName
                phoneNumber
                address
        POST
                studentId
                firstName
                lastName
                phoneNumber
                address

/api/students/{studentId}
        GET
                studentId
                firstName
                lastName
                phoneNumber
                address
        PUT
                studentId
                firstName
                lastName
                phoneNumber
                address
        DELETE
                None
```

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.

GET /api/students

```
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.

GET /api/students
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

```
        Courses
                /api/v1/courses
                /api/v1/courses/{courseId}
                /api/v1/courses/{courseId}/students
        Students
                /api/v1/students
                /api/v1/students/{studentId}
```

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/{collegeId}/**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

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**

4XX Client Error Status Codes

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

5XX Server Error Status Codes

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

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.