Students API Design Documentation

STEP 1: Create a New API

Title: OpenAPI Specification for CMS – Students API
Description: API Specification document for managing students in the CMS system
Contact: Praveenkumar Bouna (http://myorganization.com/staff/praveenkumar-bouna)
Version: 1.0

STEP 2: Identify the Type of API

Type: Public
Style: RESTful API

STEP 3: Identify the Server Base URL

http://{hostname}:{portnumber}/{directory}
http://localhost:44333

STEP 4: Identify the Resources

students

STEP 5: Use Plural Resources

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

STEP 6: Define the Resource Model

Student Model:
- studentId: int
- firstName: string
- lastName: string
- phoneNumber: string
- address: string

STEP 7: Select the Identifier

StudentId (unique identifier)

STEP 8: Associations

/api/students
/api/students/{studentId}
Related: /api/courses/{courseId}/students

STEP 9: URL Complexity

Collection/Item pattern kept simple

STEP 10: Identify Operations

/api/students → GET (all), POST (new)
/api/students/{studentId} → GET (by ID), PUT (update), DELETE (remove)

STEP 11: Parameters

Path: studentId
Query: page, size, sortBy
Headers: None

STEP 12: Content-Type of Request

application/json

STEP 13: Request Body

POST /api/students:
{
"firstName": "John",
"lastName": "Doe",
"phoneNumber": "1234567890",
"address": "New York"
}

PUT /api/students/{studentId}:
{
"firstName": "John",
"lastName": "Smith",
"phoneNumber": "9876543210",
"address": "California"
}

STEP 14: Status Codes

GET → 200 OK
POST → 201 CREATED, 400 BAD REQUEST
GET by ID → 200 OK, 404 NOT FOUND
PUT → 200 OK, 404 NOT FOUND
DELETE → 204 NO CONTENT, 404 NOT FOUND

STEP 15: Response Content-Type

application/json

STEP 16: Response Body

GET /api/students:
[
{
"studentId": 1,
"firstName": "John",
"lastName": "Doe",
"phoneNumber": "1234567890",
"address": "New York"
}
]

GET /api/students/{studentId}:
{
"studentId": 1,
"firstName": "John",
"lastName": "Doe",
"phoneNumber": "1234567890",
"address": "New York"
}

STEP 17: Error Handling

400 BAD REQUEST:
{
"error": {
"code": "INVALID_INPUT",
"message": "One or more input arguments are invalid",
"target": "Student",
"details": [
{ "code": "MISSING_FIELD", "target": "firstName", "message": "First name is required" }
]
}
}

STEP 18–20: Filtering, Pagination, Sorting

Filtering: GET /api/students?lastName=Smith
Pagination: GET /api/students?page=1&size;=20
Sorting: GET /api/students?sortBy=lastName

STEP 21: API Versioning

Scheme: URL Versioning
Version: 1.0
/api/v1/students
/api/v1/students/{studentId}