REST API Guidelines

Guidelines for designing REST API

REST API Guidelines

REST API should be designed properly as per these guidelines.

URI structure

- Serving multiple services from same API: `https://[host]:[port]/api/{service name}]/v{version number}/{resource}`
- Serving single service from same API: `https://[host]:[port]/api/v{version number}/{resource}`
- A trailing forward slash (`/`) should not be included in URIs (Mandatory)
- Forward slash separator (`/`) must be used to indicate a hierarchical relationship (Mandatory)
- Use Hyphens (`-`) instead of Underscores (`_`) if separation of words is needed in the URI.
- Lowercase letters should be preferred in URI paths.
- File extensions should not be included in URIs.
- A plural noun should be used for collection names.

Security

- Always use `HTTPS` . Never use `HTTP` it can put data exchanged on risk.
- Use token based authentication

Versioning

- When APIs are being consumed by the world, upgrading the APIs with some breaking change would also lead to breaking the existing products or services using your APIs.
- Always use versioning in API. Use convention `v[Version Number]` for versioning.
- `http://api.yourservice.com/v1/companies/34/employees` is a good example, which has the version number of the API in the path. If there is any major breaking update, we can name the new set of APIs as `v2` or `v1.x.x` .
- Use Semantic Versioning. Given a version number MAJOR.MINOR.PATCH, increment as below:

Version	Scenario

Version	Scenario
MAJOR	Incompatible API changes
MINOR	Add functionality in a backwards compatible manner
PATCH	Make backwards compatible bug fixes.

API endpoint

- The URL should only contain resources(nouns) not actions or verbs. So never use endpoints like `/addNewEmployee` , `/updateEmployee` , etc. Instead, use `/employee` and use HTTP methods to tell API which action to perform.
- The resource should always be plural in the API endpoint. So use `/users` , `/users/12` , etc. instead of `/user` , `/user/12` .

HTTP methods

Method	Usage
GET	**GET** method requests data from the resource and should not produce any side effect. So no insert, update or delete operation should be performed while serving this API. E.g /companies/3/employees returns list of all employees from company 3.
POST	**POST** method requests the server to create a resource in the database. E.g `/companies/3/employees` creates a new `Employee` of company `3` . **POST** is non-idempotent which means multiple requests will have different effects.
PUT	**PUT** method requests the server to update resource or create the resource, if it doesn't exist. E.g. `/companies/3/employees/john` will request the server to update, or create if doesn't exist, the `john` resource in `employees` collection under company `3` . PUT is idempotent which means multiple requests will have the same effects.
DELETE	**DELETE** method requests that the resources, or its instance, should be removed from the database. E.g `/companies/3/employees/john/` will request the server to delete `john` resource from the `employees` collection under the company `3` . Better to do such action on ID though, instead of name.

HTTP response status codes

2xx (Success category)

These status codes represent that the requested action was received and successfully processed by the server.

Code	Purpose
200 OK	The standard HTTP response representing success for GET, PUT or POST.
201 Created	This status code should be returned whenever the new instance is created. E.g. on creating a new instance, using `POST` method, should always return 201 status code.
204 No Content	Represents that request is successfully processed, but has not returned any content. `DELETE` can be a good example of this. The API `DELETE /companies/43/employees/2` will delete the employee `2` and in return we do not need any data in the response body of the API, as we explicitly asked the system to delete. If there is any error, like if employee `2` does not exist in the database, then the response code would not be of `2xx Success` Category but around `4xx Client Error` category.

3xx (Redirection Category)

Code	Purpose
304 Not Modified	Indicates that the client has the response already in its cache. And hence there is no need to transfer the same data again.

4xx (Client Error Category)

These status codes represent that the client has raised a faulty request.

Code	Purpose
400 Bad Request	Indicates that the request by the client was not processed, as the server could not understand what the client is asking for.
401 Unauthorized	Indicates that the client is not allowed to access resources, and should re-request with the required credentials.
403 Forbidden	Indicates that the request is valid and the client is authenticated, but the client is not allowed access the page or resource for any reason. E.g sometimes the authorized client is not allowed to access the directory on the server.
404 Not Found	Indicates that the requested resource is not available now.
405 Method not allowed	Indicates that the client tried to use an HTTP method that the resource does not allow. For instance, a read-only resource could support only `GET` , but not `POST` , `PUT` or `DELETE`

Code	Purpose
410 Gone	Indicates that the requested resource is no longer available which has been intentionally moved.

5xx (Server Error Category)

Code	Purpose
500 Internal Server Error	Indicates that the request is valid, but the server is totally confused and the server is asked to serve some unexpected condition.
503 Service Unavailable	Indicates that the server is down or unavailable to receive and process the request. Mostly if the server is undergoing maintenance.

Error Payloads

All exceptions should be mapped in an error payload. Here is an example how a JSON payload should look like.

```
{
"status": 404,
"message": ""Sorry, the requested resource does not exist"",
"code": 34,
"more_info": "https://api.service.com/docs/errors/34 "
}
```

Field name casing convention

Follow camelCase to maintain the consistency.

Sorting

In case, the client wants to get the sorted list of companies, the `GET /companies` endpoint should accept multiple sort params in the query.
E.g. `GET /companies?sort=rank_asc` would sort the companies by its rank in ascending order.

Filtering

For filtering the dataset, we can pass various options through query params.
E.g. `GET /companies?category=banking&location=india` would filter the companies list data with the company category of Banking and where the location is India.

Searching

When searching the company name in companies list, the API endpoint should be `GET /companies?search=Digital Mckinsey`

Pagination

When the dataset is too large, we divide the data set into smaller chunks, which helps in improving the performance and is easier to handle the response.
E.g. `GET /companies?page=23` means get the list of companies on 23rd page.