RESTful API Designing guidelines – The best practices

Originally published by Mahesh Haldar on March 25th 2016 525,811 reads

Start Writing Log in

.tech DOMAINS Never get a boring domain. Get .Tech!

Mahesh Haldar

Facebook, Google, Github, Netflix and few other tech giants have given a chance to the developers and products to consume their data through APIs, and became a platform for them.
Even if you are not writing APIs for other developers and products, it is always very healthy for your application to have beautifully crafted APIs.

There is a long debate going on the internet, about the best ways to design the APIs, and is one of the most nuanced. There are no official guidelines defined for the same.

The API is an interface, through which many developers interact with the data. A good designed API is always very easy to use and makes the developer's life very smooth. API is the GUI for developers, if it is confusing or not verbose, then the developer will start finding the alternatives or stop using it. Developers' experience is the most important metric to measure the quality of the APIs.

The API is like an artist performing on stage, and its users are the audience

1) Terminologies

The following are the most important terms related to REST APIs

- **Resource** is an object or representation of something, which has some associated data with it and there can be set of methods to operate on it. E.g. Animals, schools and employees are resources and *delete, add, update* are the operations to be performed on these resources.
- **Collections** are set of resources, e.g *Companies* is the collection of *Company* resource.
- **URL** (Uniform **Resource** Locator) is a path through which a **resource** can be located and some actions can be performed on it.

2) API endpoint

Let's write few APIs for **Companies** which has some **Employees,** to understand more.
`/getAllEmployees` is an API which will respond with the list of employees. Few more APIs around a **Company** will look like as follows:

- *`/addNewEmployee`*
- *`/updateEmployee`*
- *`/deleteEmployee`*
- *`/deleteAllEmployees`*
- *`/promoteEmployee`*
- *`/promoteAllEmployees`*

And there will be tons of other API endpoints like these for different operations. All of these will contain many redundant

And there will be tons of other API endpoints like these for different operations. All of those will contain many redundant actions. Hence, all these API endpoints would be burdensome to maintain, when API count increases.

What is wrong?

The URL should only contain resources(nouns) not actions or verbs. The API path`/addNewEmployee` contains the action `addNew` along with the resource name `Employee`.

Then what is the correct way?

`/companies` endpoint is a good example, which contains no action. But the question is how do we tell the server about the actions to be performed on `companies` resource viz. whether to add, delete or update?

This is where the HTTP methods (GET, POST, DELETE, PUT), also called as verbs, play the role.

The resource should always be **plural** in the API endpoint and if we want to access one instance of the resource, we can always pass the id in the URL.

- method `GET` path `/companies` should get the list of all companies
- method `GET` path `/companies/34` should get the detail of company 34
- method `DELETE` path `/companies/34` should delete company 34

In few other use cases, if we have resources under a resource, e.g Employees of a Company, then few of the sample API endpoints would be:

- `GET /companies/3/employees` should get the list of all employees from company 3
- `GET /companies/3/employees/45` should get the details of employee 45, which belongs to company 3

- `DELETE /companies/3/employees/45` should delete employee 45, which belongs to company 3
- `POST /companies` should create a new company and return the details of the new company created

Isn't the APIs are now more precise and consistent? 😎

Conclusion: *The paths should contain the plural form of resources and the HTTP method should define the kind of action to be performed on the resource.*

3) HTTP methods (verbs)

HTTP has defined few sets of methods which indicates the type of action to be performed on the resources.

The URL is a sentence, where resources are nouns and HTTP methods are verbs.

The important HTTP methods are as follows:

1. `GET` method requests data from the resource and should not produce any side effect.
   E.g `/companies/3/employees` returns list of all employees from company 3.
2. `POST` method requests the server to **create** a resource in the database, mostly when a web form is submitted.
   E.g `/companies/3/employees` creates a new Employee of company 3.
   `POST` is non-idempotent which means multiple requests will have different effects.

3. `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.

4. `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.

There are few other methods which we will discuss in another post.

4) HTTP response status codes

When the client raises a request to the server through an API, the client should know the feedback, whether it failed, passed or the request was wrong. HTTP status codes are bunch of standardized codes which has various explanations in various scenarios. The server should always return the right status code.
The following are the important categorization of HTTP codes:

2xx (Success category)

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

- **200 Ok** The standard HTTP response representing success for GET, PUT or POST

- **200 OK** The standard HTTP response representing success for GET, PUT or PATCH.
- **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 the 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 be not be of `2xx Success Category` but around `4xx Client Error category`.

3xx (Redirection Category)

- **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.

- **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.

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.
- **410 Gone** indicates that the requested resource is no longer available which has been intentionally moved.

5xx (Server Error Category)

- **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.

5) Field name casing convention

You can follow any casing convention, but make sure it is consistent across the application. If the request body or response type is JSON then please follow camelCase to maintain the consistency.

6) Searching, sorting, filtering and pagination

All of these actions are simply the query on one dataset. There will be no new set of APIs to handle these actions. We need to append the query params with the GET method API

append the query params with the GET method API.

Let's understand with few examples how to implement these actions.

- **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. Eg. `GET /companies?page=23` means get the list of companies on 23rd page.

If adding many query params in GET methods makes the URI too long, the server may respond with `414 URI Too long` HTTP status, in those cases params can also be passed in the request body of the `POST` method.

7) Versioning

When your 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.

`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`

These guidelines are compiled on my experience of development. I would love to know your views on the pointers mentioned above. Please leave a comment, and let me know!

If this article helped you, then you can buy me a coffee 😊

1

Share this story

@haldar.mahesh
Mahesh Haldar

Read my stories

RELATED

Learn how to gain API performance visibility today

Visit SmartBear Software promoted

https://bit.ly/3g8JRF6

RT-Thread Studio IDE Upgrades into v2.0.0 to Support More Kinds of ...

pre-emoji story

@RT-Thread IoT OS

RT-Thread IoT OS 12/13/20

How To Setup Continuous Integration Pipeline By Using Terraform And...

pre-emoji story

@EmmanuelSys

Emmanuel Sys 12/13/20

TAGS

#api #restful #best-practices #software-development #web-development

Join Hacker Noon

Help **About** **Start Writing** **Sponsor:** Brand-as-Author Sitewide Billboard Ad by tag Newsletter Noonies
Contact Us Terms Privacy Cookies Stories published yesterday Leaderboard Contributors' Club Chrome Extension