**REST verbs**

* REST verbs specify an action to be performed on a specific resource or a collection of resources. When a request is made by the client, it should send this information in the HTTP request.
* RESTful APIs primarily use HTTP verbs (also known as HTTP methods or REST verbs) to indicate the desired action to be performed on a resource.
* REST uses the URI to decode its resource to be handled. There are quite a few REST verbs available, but six of them are used frequently.
* They are as follows:
* GET
* POST
* PUT
* PATCH
* DELETE
* OPTIONS

What is URI?

* URI stands for Uniform Resource Identifier. It is a string of characters that identifies a particular resource or group of resources on the internet. A URI provides a unique way to identify and locate resources, such as web pages, images, documents, APIs, and more.

The following table explains the operation, target resource, and what happens if the request succeeds or fails:

|  |  |  |  |
| --- | --- | --- | --- |
| **REST Verb** | **Action** | **Success** | **Failure** |
| GET | Fetches a record or set of resources from the server | 200 | 404 |
| OPTIONS | Fetches all available REST operations | 200 | – |
| POST | Creates a new set of resources or a resource | 201 | 404, 409 |
| PUT | Updates or replaces the given record | 200, 204 | 404 |
| PATCH | Modifies the given record | 200, 204 | 404 |
| DELETE | Deletes the given resource | 200 | 404 |

The numbers in the Success and Failure columns of the preceding table are HTTP status codes. Whenever a client initiates a REST operation, since REST is stateless, the client should know a way to find out whether the operation was successful or not. For that reason, HTTP has status codes for the response. REST defines the preceding status code types for a given operation. This means a REST API should strictly follow the preceding rules to achieve client-server communication.

All defined REST services have the following format. It consists of the host and API endpoint. The API endpoint is the URL path which is predefined by the server.

REST API URI**: http://HostName/API endpoint/Query(optional)**

**GET:**

The GET method is used to retrieve a representation of a resource or a collection of resources from the server. It should not have any side effects on the server and is primarily used for retrieving data.

GET uses a few types of URI queries:

* Query parameters
* Path-based parameters

Query Parameters:

* Query parameters are appended to the URL after a question mark (?) and separated by ampersands (&).
* They are used to provide additional data to the server for filtering, sorting, searching, or any other custom parameters.
* Query parameters do not affect the structure of the URL path and are optional.
* Example: GET /api/users?role=admin&status=active
* In this example, the query parameters role=admin and status=active are added to the URL. The server can use these parameters to filter and retrieve users with the specified role and status.

Path-based Parameters:

* Path-based parameters are included directly in the URL path itself, usually indicated by a placeholder.
* They are used to specify a specific resource or identify a unique entity within the URL hierarchy.
* Path-based parameters are often used for identifying a specific resource by its unique identifier, such as an ID or a slug.
* Example: GET /api/users/123
* In this example, the path-based parameter 123 is used to identify the specific user with the ID 123. The server can retrieve the user resource associated with this ID.

Both query parameters and path-based parameters can be used in combination to provide more specific and flexible data retrieval

Example:

GET /api/users/123

This example retrieves the user with ID 123 from the /api/users endpoint.

**POST:**

The POST method is used to submit data to the server to create a new resource. It typically includes a payload in the request body that will be processed by the server to create the resource.

It is not idempotent, as each request typically creates a new resource.

Example:

POST /api/users

Content-Type: application/json

{

"name": "John Doe",

"email": "johndoe@example.com"

}

This example creates a new user by sending user data in the request body to the /api/users endpoint.

**PUT:**

The PUT method is used to update or replace a resource on the server. It requires sending the complete representation of the resource in the request body, which will replace the existing resource.

Example:

PUT /api/users/123

Content-Type: application/json

{

"name": "John Doe Jr.",

"email": "john.doe@example.com"

}

This example updates the user with ID 123 by sending the updated user data in the request body to the /api/users/123 endpoint.

**PATCH:**

The PATCH method is used to partially update a resource on the server. It is similar to the PUT method but allows sending only the changes or modifications to the resource instead of the complete representation.

Example:

PATCH /api/users/123

Content-Type: application/json

{

"email": "newemail@example.com"

}

This example applies partial updates to the user with ID 123 by sending the modified fields in the request body to the /api/users/123 endpoint.

**DELETE:**

The DELETE method is used to delete a resource from the server. It instructs the server to remove the specified resource.

Example:

DELETE /api/users/123

This example deletes the user with ID 123 by sending a DELETE request to the /api/users/123 endpoint.