**REST API**

**Re**presentational **S**tate **T**ransfer: REST provides certain guidelines which client and server mus follow while transferring the data.  
Whenever a client requests any data. Server converts the information (from its current state) into a format on which client and server have agreed to perform the exchange.  
Then the data is exchanged between client and server in the decided format.

**Guidelines for creating RESTful APIs**

1. It works in **client-server** architecture. Client requests data, server responds with data.
2. It checks if data from server is **cache**able or not cacheable. If Cacheable, UI can store and reuse that data again.
3. **Layered** architecture should be present. Client requesting the data should only know the immediate next layer (that is server endpoint), it should not know which service or database is called to get that data.
4. **Stateless.** One server can get request from multiple clients. If C1 requested for getCount() and server responded with 4 as answer. After sometime, C2 requested for getCount( ), then server should treat C2 as new client and fetch getCount( ) data again from services/DB. It should not store the response which it sent for C1 and send the same to C2. This is not allowed. This is called being stateless. Where we do ot store the state of response. Every new request is treated uniformly and newly.
5. The **interface** for request and response should be **uniform**. It should not happen that one API is handled over HTTP and other is handled over some other protocol.
6. **Code on Demand** (optional): This means that server can send code to client, which it needs to execute in runtime. Ex: Java Applets / Javascript.

Example of REST API:

Suppose we are designing a Book Store

Following capabilities required:

1. **Get list of all books**

**Request**

HTTP Method: GET

<https://api.allaboutbooks.com> /myStore/books 🡨--- Resource Path

Domain Name

**Response Data**:

{ [

{id: 1, name: ABC},

{id: 2, name: DEF},

{id: 3, name: PQR},{….}

]}

1. **Add new book in store**

**Request**

HTTP Method: POST

<https://api.allaboutbooks.com> /myStore/book

Payload: {Id: 4, Name: XYZ, Count = 100}

**Response**

{id: 4, Status Code: 200 }

1. **Update count of a book**

**Request**

HTTP Method: PUT

<https://api.allaboutbooks.com> /myStore/book/2

Payload: {Count = 3}

**Response**

{id: 2, Status Code: 200 }

1. **Delete a book**

**Request**

HTTP Method: DELETE

<https://api.allaboutbooks.com> /myStore/book/2

**Response**

{id: 2, Status Code: 200 }

Path parameter

Ex: <https://api.allaboutbooks.com/myStore/book/2>

Here, in the URI, we have specified ID of the book we are looking for “2”.

This is called path parameter.

Query Parameter

Suppose we don’t want to show all books at once on our webpage. We have used pagination and need to show 20 books per page.

Then URI can be modified as <https://api.allaboutbooks.com/myStore/books?limit=20&offset=0>

This will show only first 20 books present in DB.

Next 20 books on page 2: <https://api.allaboutbooks.com/myStore/books?limit=20&offset=21>

This means, starting from 21st row, show 20 books

Next 20 books on page 3:

<https://api.allaboutbooks.com/myStore/books?limit=20&offset=41>

and so on…

Suppose I want all fiction books only:

<https://api.allaboutbooks.com/myStore/books?tags=fiction>

This will give us all books who’s tag data member/ column has “fiction”

**HTTP Response**

**HTTP Status Codes:**

**2xx**: Success codes 🡪Update, create, delete, fetch is done successfully.

**3xx**: Redirection 🡪the resource you are trying to fetch is not at this location, it is now at some other location.

**4xx**: Client Error 🡪 Client is making error in making request / Bad Request/ or

409: Client is trying to create a resource which already exist (Conflict).

412: it is sending data on which client-server have not agreed.

**5xx**: Server Error 🡪 There is some problem at server side, in processing the request. (Internal Server Error)

**Security**

There could be misuse of APIs. To protect the data, we can

1. Safeguard the implementation of the API and validate the data
2. Propagate errors properly and generate logs.
3. Secure the APIs by authorization tokens (private APIs) – Oauth2 and JWT.
4. Rate limiting and Throttling.