### **Document: Understanding REST APIs and Common Commands Used to Work with Them**

#### **1. Introduction to REST APIs**

**REST** (Representational State Transfer) is an architectural style for designing networked applications. It is based on a set of principles and constraints that allow web services to be lightweight, scalable, and easy to use. REST APIs enable interaction between clients and servers over the HTTP protocol.

A **REST API** (RESTful API) is a set of web service APIs that follow the REST architecture principles. They allow clients (such as web browsers, mobile apps, or other systems) to interact with the server using HTTP methods like GET, POST, PUT, DELETE, and PATCH to perform CRUD operations (Create, Read, Update, Delete) on resources.

#### **2. REST API Principles**

* **Stateless**: Each request from a client to a server must contain all the information the server needs to fulfill the request. The server does not store any state between requests.
* **Client-Server Architecture**: The client and server are separate entities, allowing them to be developed, deployed, and maintained independently.
* **Uniform Interface**: REST APIs have a consistent and standard structure for interacting with resources.
* **Cacheable**: Responses from the server must be explicitly labeled as cacheable or non-cacheable to improve performance.
* **Layered System**: The architecture can have multiple layers between the client and server, improving scalability and security.
* **Code on Demand (optional)**: Servers can provide executable code (e.g., JavaScript) to clients, but this is an optional constraint.

#### **3. Common HTTP Methods Used in REST APIs**

RESTful APIs use standard HTTP methods to interact with resources on the server:

* **GET**: Retrieves data from the server. It does not modify any data and is idempotent (multiple requests result in the same response).
  + Example: GET /users retrieves a list of users.
* **POST**: Sends data to the server to create a new resource. It may return the created resource or a status.
  + Example: POST /users creates a new user.
* **PUT**: Updates an existing resource or creates a new one if it doesn’t exist. It is idempotent, meaning that sending the same request multiple times will have the same effect.
  + Example: PUT /users/1 updates the user with ID 1.
* **DELETE**: Deletes a resource from the server.
  + Example: DELETE /users/1 deletes the user with ID 1.
* **PATCH**: Partially updates an existing resource. It is generally used when updating a few fields of a resource.
  + Example: PATCH /users/1 updates a specific field of user ID 1.

#### **4. REST API Example**

Suppose we have a REST API for managing users. The base URL for the API is <https://api.example.com/users>. Below are examples of how the various HTTP methods are used with this API.

* **GET**: Retrieve all users

GET <https://api.example.com/users>

* **GET**: Retrieve a specific user by ID

GET <https://api.example.com/users/1>

* **POST**: Create a new user

POST <https://api.example.com/users>Body:  
{  
 "name": "John Doe",  
 "email": "[john@example.com](mailto:john@example.com)",  
 "age": 30  
}

* **PUT**: Update an existing user

PUT <https://api.example.com/users/1>Body:  
{  
 "name": "John Updated",  
 "email": "[john.updated@example.com](mailto:john.updated@example.com)",  
 "age": 31  
}

* **DELETE**: Delete a specific user

DELETE <https://api.example.com/users/1>

* **PATCH**: Partially update a user's information

PATCH <https://api.example.com/users/1>Body:  
{  
 "email": "[john.updatedagain@example.com](mailto:john.updatedagain@example.com)"  
}

#### **5. REST API Commands and Tools**

Here are some commands and tools used to interact with REST APIs:

1. **cURL (Command Line Tool)**:
   1. cURL is a command-line tool used to make HTTP requests to APIs.
   2. Example of making a GET request using cURL:

curl -X GET <https://api.example.com/users>

* 1. Example of making a POST request using cURL:

curl -X POST <https://api.example.com/users> -d '{"name": "John", "email": "[john@example.com](mailto:john@example.com)"}' -H "Content-Type: application/json"

1. **Postman**:
   1. Postman is a graphical user interface (GUI) tool that helps you interact with REST APIs, making it easier to send requests and view responses.
   2. You can define the HTTP method (GET, POST, etc.), set headers, and include request bodies (for POST, PUT, PATCH).
2. **HTTPie**:
   1. HTTPie is a command-line HTTP client that is user-friendly and makes working with APIs easier than cURL.
   2. Example of making a GET request using HTTPie:

http GET <https://api.example.com/users>

1. **PowerShell (Invoke-RestMethod)**:
   1. PowerShell has built-in cmdlets like Invoke-RestMethod to interact with REST APIs.
   2. Example of making a GET request using PowerShell:

Invoke-RestMethod -Uri "<https://api.example.com/users>" -Method Get

* 1. Example of making a POST request using PowerShell:

$body = '{"name": "John", "email": "[john@example.com](mailto:john@example.com)"}'  
Invoke-RestMethod -Uri "<https://api.example.com/users>" -Method Post -Body $body -ContentType "application/json"

#### **6. Authentication in REST APIs**

Most APIs require authentication to ensure that the user or system making the request has permission to access the resources. Common authentication methods include:

1. **API Key**:
   1. An API key is a unique identifier passed along with requests to authenticate the user.
   2. Example (using cURL):

curl -X GET <https://api.example.com/users> -H "API-Key: your\_api\_key"

1. **Basic Authentication**:
   1. This method involves passing a username and password encoded in base64.
   2. Example (using cURL):

curl -u "username:password" -X GET <https://api.example.com/users>

1. **Bearer Token (OAuth)**:
   1. Bearer tokens are used with OAuth 2.0 for authentication. The token is passed in the Authorization header.
   2. Example (using cURL):

curl -X GET <https://api.example.com/users> -H "Authorization: Bearer your\_bearer\_token"

#### **7. REST API Response Codes**

API responses typically include an HTTP status code indicating the result of the request:

* **200 OK**: The request was successful.
* **201 Created**: A new resource was successfully created.
* **400 Bad Request**: The request was malformed or contains invalid data.
* **401 Unauthorized**: Authentication failed or was not provided.
* **403 Forbidden**: The server understood the request, but it refuses to authorize it.
* **404 Not Found**: The requested resource could not be found.
* **500 Internal Server Error**: A generic error occurred on the server.

#### **8. Conclusion**

REST APIs are widely used for enabling communication between different systems over the web. Understanding how to interact with them using HTTP methods like GET, POST, PUT, DELETE, and PATCH is essential for building and consuming web services. Tools like cURL, Postman, and PowerShell make it easier to work with REST APIs, and knowing how to handle authentication, error codes, and status responses is crucial for developing and consuming APIs effectively.