REST API

Abdullah Al Asif
Associate Software Engineer
CEFALO Bangladesh LTD

Table of Contents

Introduction to RESTful API

- What is REST API
- History of REST API
- Principles

What is REST API?

A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data.

REST API History (Brief)

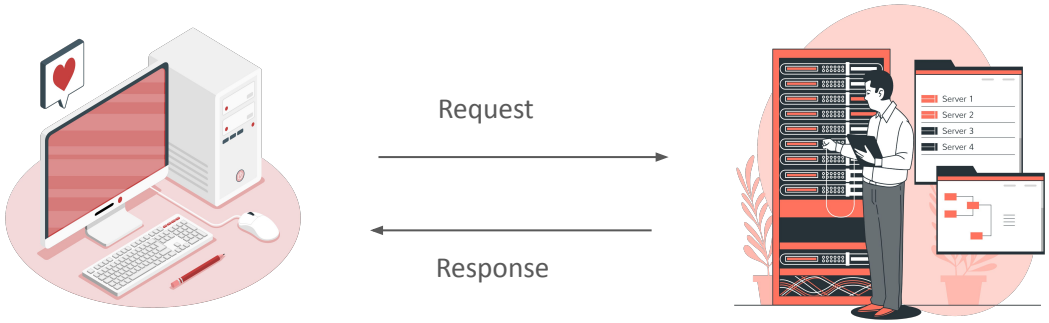
- 1990s: HTTP introduced for the web.
- 2000: Roy Fielding's REST dissertation.
- Key Principles: URIs, statelessness, HTTP methods.
- Adoption: Simplicity, major companies.
- HTTP Methods: GET, POST, PUT, DELETE.

REST Principles

- Client–server Architecture
- Uniform interface
- Stateless
- Cacheable
- Layered system
- Code on demand (optional)

Client-Server Architecture

Separation of the client (user interface and user experience concerns) and the server (data storage and business logic), allowing for independent development and scalability



Statelessness

Each request from a client to a server must contain all the information needed to understand and fulfill the request. The server should not store any client state between requests.

Uniform Interface

A consistent and standardized way to interact with the API, promoting simplicity and predictability. It includes resource identification, manipulation through representations, and self-descriptive messages

Cacheability

Responses should indicate whether they can be cached or not. Caching helps improve performance by allowing clients to reuse previously retrieved responses.

Layered System

The architecture can be composed of multiple layers, such as load balancers, security protocols, and databases. Each layer has a specific role, promoting flexibility and scalability.

Code on Demand (Optional)

Servers can optionally send executable code to clients. This feature is rarely used in practice and is considered optional for a RESTful API.

Creating and Consuming Restful API

- Api design
- Endpoint
- HTTP Methods

API Design

- Design the API with a clear structure and organization.
    - Imagine a library
    - Consistent Naming
    - Versioning

```
https://api.example.com/v1/products
https://api.example.com/v1/users/asif
https://api.example.com/v2/products
```

- Define how client will interact with the server.
    - Clear Instructions
    - Authentication
    - Error handling
    - Consistency

Endpoint

- Endpoints are specific paths or URLs that represent `resources`.

endpoint

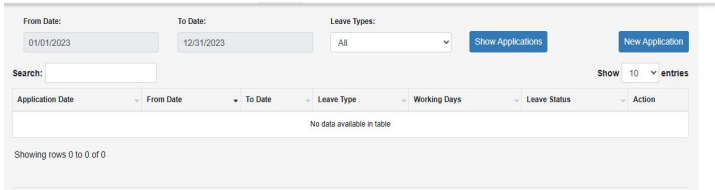
`https://hrportal.cefalolab.com/user-profile`

url

- Each endpoint corresponds to a specific function or operation.

`https://hrportal.cefalolab.com/profile/1395`

`https://hrportal.cefalolab.com/leave-application`



HTTP Methods

Different methods perform different operations:

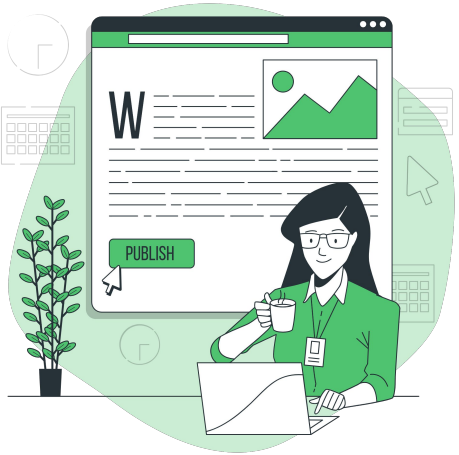
- GET: Retrieve data.
- POST: Create new data.
- PUT: Update existing data.
- DELETE: Remove data.
- ...and more.

POST

```
POST /article HTTP/1.1
Accept: application/json
```

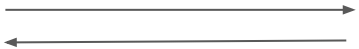
```
HTTP/1.1 201 OK
```

```
Json:
{
   “user”: “Alexa”,
   “article”: “sharing my ideas”,
}
```



GET

```
GET /cefalo/members HTTP/1.1
Accept: application/json
```

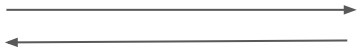


```
HTTP/1.1 200 OK
```

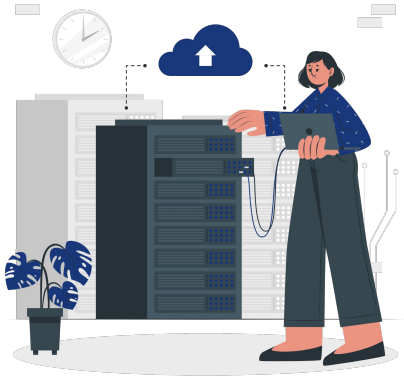
```
Json:
{
   “members”: [
       “Ferdous Mahmud Shaon”,
       “Asif Kamal”,
       “Simanta Deb Turja”,
       “Ranak”
       “Asif Abdullah”,

    ]
}
```

PUT

```
PUT /users/123 HTTP/1.1
Content-type: application/json
```



```
HTTP/1.1 204 OK
```



```
Json:
{
   “id”: 123,
   “name”: “Asif Abdullah”
   “address”: “Dhaka,Bangladesh”
}
```

DELETE

DELETE /users/123 HTTP/1.1

HTTP/1.1 204 OK

Best practices for REST API

- Descriptive and Intuitive Endpoint Names

```
GET /products → Good
GET /product/{id} → Good

GET /data → Bad
GET /data/{id} → Bad
```

- Use noun not verbs

```
GET /cefalo.com/members → Good

GET /cefalo.com/getAllMembers → Bad
```

Best practices for REST API

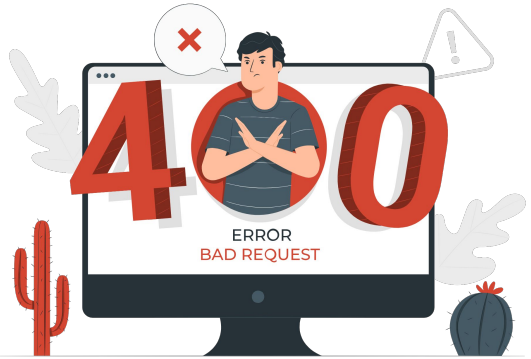
- Ensuring Security
  - Authentication → Verify the identity of user
  - Use method likes `Oauth, JWT, API keys`
  - Authorization → Define and enforce access control

- Versioning
  - Use version number for api to manage changes and updates

```
GET /v1/products
GET /v1/product/{id}
GET /v2/users
GET /v2/users/{id}
```

Best practices for REST API

- Error message
  - Descriptive error messages, maintaining consistency.



Successful responses(200-299)

200- OK
201- Created
202- Accepted
204- No content

Server error response(500-599)

500- Internal Server Error
502- Bad Gateway
503- Service Unavailable
504- Gateway timeout

Client error response(400-499)

400- Bad Request
401- Unauthorized
403- Forbidden
404- Not Found

Security of REST API

What is Authentication?

- Authentication verifies the identity of users or systems accessing the API.

We will discuss about 3 techniques

JWT, OAuth, API Key

OAuth

OAuth is *not* an API or a service: it's an open standard for authorization and anyone can implement it. It is useful for accessing third-party applications without sharing your passwords.

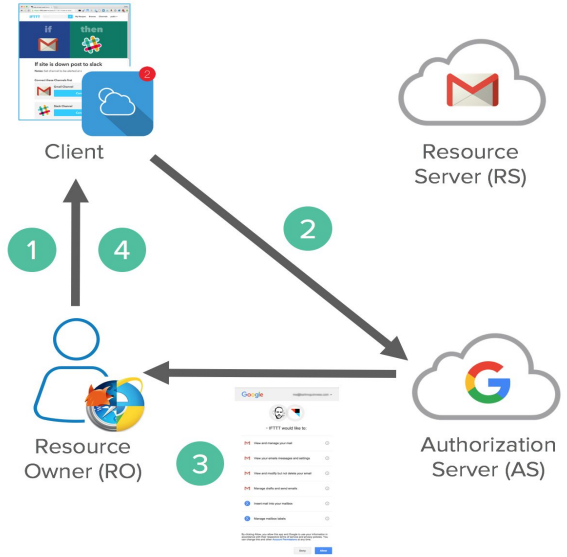
Benefits

- Enhanced security by avoiding password sharing.
- Enables seamless user authentication across platforms.

How it works

Main Components:

- Scopes and Consent
- Actors
- Clients
- Tokens
- Authorization Server
- Flows



Request

1

```
GET https://accounts.google.com/o/oauth2/auth?
scope=gmail.insert gmail.send
&redirect_uri=https://app.example.com/oauth2/callback
&response_type=code&client_id=812741506391
&state=af0ifjsldkj
```

Response

2

```
HTTP/1.1 302 Found
Location: https://app.example.com/oauth2/callback?
code=MsCeLvIaQm6bTrgtp7&state=af0ifjsldkj
```

Request

3

```
POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=MsCeLvIaQm6bTrgtp7&client_id=812741506391&client_secret=
{asTWlkeg2A4gkLs4}&redirect_uri=https://app.example.com/oauth2
/callback&grant_type=authorization_code
```

Response

4

```
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

JWT(JSON Web Token)

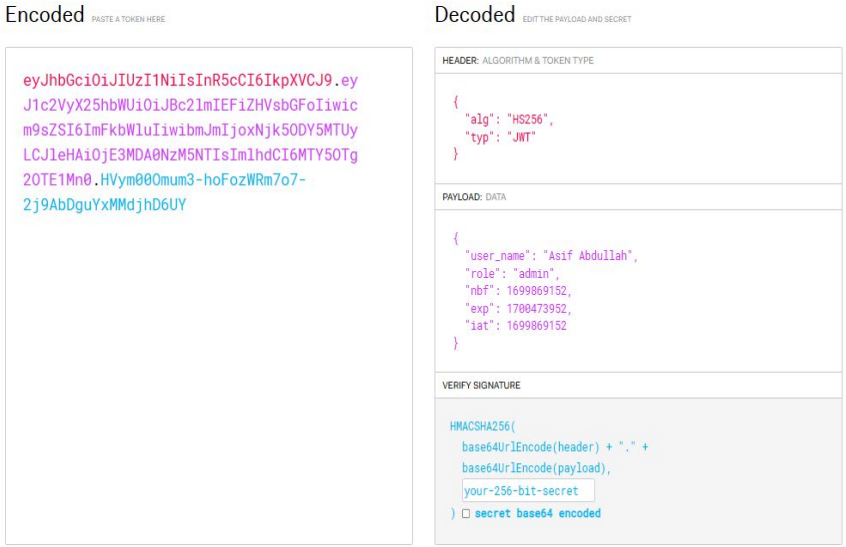
JWTs are compact, self-contained tokens for securely transmitting information between parties.

Components

- Header
- Payload
- Signature

Benefits

- Enhanced security by avoiding password sharing.
- Enables seamless user authentication across platforms.



API Keys

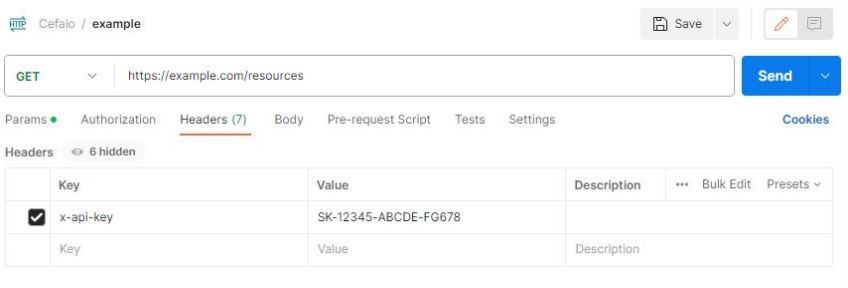
- API keys are unique strings passed by clients to the server to authenticate their identity.

- Clients include API keys in their requests, allowing servers to identify and authorize requests

Benefits

- Simple and straightforward authentication method.
- Effective for controlling access to APIs.

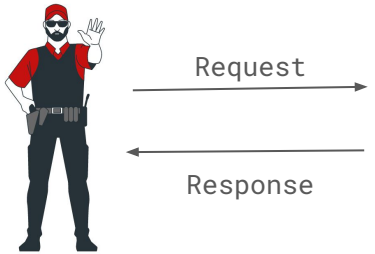
Some Examples : `a1b2c3d4e5f6g7h8` , `SK-12345-ABCDE-FG678`



CORS

- CORS stands for Cross-Origin Resource Sharing, a security feature implemented by web browsers.

- It's a security measure to control how web browsers allow one website to access resources from another website.



How it works

1

Request

```
OPTIONS /widgets/ HTTP/1.1
Host: api.mydomain.com
Origin: https://www.mydomain.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Authorization,
Content-Type
[Rest of request...]
```

2

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin:
https://www.mydomain.com
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: Authorization,
Content-Type
Content-Type: application/json
[Rest of response...]
```

3

Request

```
POST /widgets/ HTTP/1.1
Host: api.mydomain.com
Authorization: 1234567
Content-Type: application/json
Origin: https://www.mydomain.com
[Rest of request...]
```

4

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin:
https://www.mydomain.com
Content-Type: application/json
[Rest of response...]
```

Summary

- REST Principles(Client-Server, stateless, layered)
- Creating and Consuming Restful API
  - Api design
  - Endpoint (diff resources)
  - HTTP Methods (GET, POST, PUT, DELETE)
- API Authentication and Authorization
  - OAuth, JWT, API keys
- CORS


THANK
YOU!!