Web Services

Table of Contents

1. What are Web Services?

An API (Application Programming Interface) is a set of rules and protocols that allows one software application to interact with another. It defines how requests should be made, what data formats to use, and how responses should be handled. APIs enable different systems, applications, or services to communicate and share data seamlessly. In simpler terms, an API is like a bridge that connects different software components, allowing them to work together efficiently.

2. Basics of REST API

Definition of REST

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building and interacting with web services. REST APIs use HTTP methods (such as GET, POST, PUT, DELETE) to perform operations on resources, which are typically represented in formats like JSON or XML. These APIs are stateless, meaning each request from a client to the server must contain all the information needed to understand and process the request. RESTful services are designed to be scalable, flexible, and easy to use, making them widely adopted in web development.

3. REST vs. HTTP

- REST is an architectural style for designing web services, while HTTP is a protocol used for communication on the web.
- RESTful services are commonly implemented using HTTP, but REST can, in theory, be implemented over any protocol.
- REST defines how resources should be accessed and manipulated, while HTTP provides the methods (GET, POST, etc.) to perform those operations.
- In summary, REST is a set of principles for web services, and HTTP is the protocol that is often used to implement those principles.

4. HTTP Methods

- GET
- POST
- PUT
- DELETE
- PATCH

4.1. GET

The `GET` method is used to retrieve information from the server. It is a read-only operation and does not alter any data on the server.

```
GET /users/123 HTTP/1.1
Host: example.com
```

Response:

```
{
    "id": 123,
    "name": "John Doe",
    "email": "john.doe@example.com"
}
```

4.2. POST

The `POST` method is used to create a new resource on the server. It sends data to the server to create a new entry.

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Jane Smith",
    "email": "jane.smith@example.com"
}
```

Response:

```
{
    "id": 124,
    "name": "Jane Smith",
    "email": "jane.smith@example.com"
}
```

4.3. PUT

The `PUT` method is used to update an existing resource on the server. It replaces the current representation of the resource with the new data.

```
PUT /users/123 HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "John Doe",
    "email": "john.new@example.com"
}
```

Response:

```
{
    "id": 123,
    "name": "John Doe",
    "email": "john.new@example.com"
}
```

4.4. DELETE

The `DELETE` method is used to delete a resource from the server.

```
DELETE /users/123 HTTP/1.1
```

```
Host: example.com
```

Response:

```
{
    "message": "User deleted successfully"
}
```

4.5. PATCH

The `PATCH` method is used to apply partial modifications to a resource. It updates only the specified fields of the resource.

```
PATCH /users/123 HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "email": "john.updated@example.com"
}
```

Response:

```
{
    "id": 123,
    "name": "John Doe",
    "email": "john.updated@example.com"
}
```

This overview provides a brief description and examples of how to use the common HTTP methods in RESTful web services to perform operations on resources.

- Status Codes
- 1xx (Informational)
- 2xx (Success)
- 3xx (Redirection)
- 4xx (Client Error)
- 5xx (Server Error)

5. Anatomy of API Request

An API request is a call made by a client to a server, asking the server to perform a specific action

or return data. The structure of an API request typically includes the following components:

1. Endpoint (URL):

- The endpoint is the URL that specifies the location of the API resource.
- It typically includes the base URL (domain) and the path to the specific resource.
- Example: `https://api.example.com/users/123`.

2. HTTP Method (Verb):

- This defines the action to be performed on the resource.
- Common methods include:
  - GET: Retrieve data.
  - POST: Create a new resource.
  - PUT: Update an existing resource.
  - DELETE: Remove a resource.

3. Headers:

- Headers provide additional information about the request.
- Common headers include:
  - `Content-Type`: Specifies the format of the request body (e.g., `application/json`).
  - `Authorization`: Includes credentials for authentication (e.g., tokens).

4. Query Parameters:

- Optional parameters appended to the URL to filter or modify the request.
- Example: `https://api.example.com/users?status=active` (filters users by active status).

5. Request Body:

- Used in methods like POST and PUT to send data to the server.
- The body usually contains the resource data in a format like JSON.
- Example:

```
{
"name": "John Doe",
"email": "john.doe@example.com"
}
```

6. Authentication/Authorization: - Ensures that the client has the right permissions to access or modify the resource. - This is often handled via tokens, API keys, or OAuth.

Example API Request:

```
POST /users HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer <token>

{
    "name": "John Doe",
    "email": "john.doe@example.com"
}
```

In this example: - The endpoint is `/users`. - The method is `POST`. - The headers include `Content-Type` and `Authorization`. - The request body contains the data for the new user.

This structured approach allows the client and server to communicate effectively and securely in a RESTful manner.

6. Consuming RESTful APIs

- Tools for Testing APIs (Postman, curl)
- Making Requests (with Python, JavaScript)
- Parsing Responses

7. Error Handling

- Client-Side Errors
- Server-Side Errors
- Error Messages and Codes

8. Practical Examples

- Building a Simple REST API with Flask (Python)
- Building a Simple REST API with Express (JavaScript/Node.js)

9. Path Parameter V/s Query Parameter

9.1. Path Parameter

- Path parameters are part of the URL path and are used to identify a specific resource.
- They are typically used to specify the ID of a resource or some other key information.

Example:

- URL: `https://api.example.com/users/123`
- Here, `123` is a path parameter that identifies a specific user.

Usage in an API request:

```
GET /users/123 HTTP/1.1
Host: api.example.com
```

In this example, the client is requesting information about the user with ID `123`.

9.2. Query Parameter

- Query parameters are appended to the end of the URL after a `?` and are used to filter or modify the request.
- They are typically used for optional parameters, such as search criteria, pagination, or sorting.

Example: - URL: `https://api.example.com/users?status=active&sort=desc` - Here, `status=active` and `sort=desc` are query parameters.

Usage in an API request:

```
GET /users?status=active&sort=desc HTTP/1.1
Host: api.example.com
```

In this example: - `status=active` filters the users to only return those who are active. - `sort=desc` sorts the results in descending order.

Summary:

- **Path parameters** are part of the URL path and are often used to uniquely identify a resource.
- **Query parameters** are added to the URL after the `?` and are used to filter, modify, or refine the request.

Here are 10 basic objective Java interview questions on REST API:

10. Question Set

What does REST stand for in the context of web services?

▼ *Click Here For Answer*

Representational State Transfer

Which HTTP method is typically used to retrieve a resource from a REST API?

▼ *Click Here For Answer*

GET

Which of the following is NOT an HTTP method used in REST APIs? A) POST B) GET C) INSERT D) DELETE

▼ *Click Here For Answer*

C) INSERT

What is the status code returned by a REST API when a resource is successfully created?

▼ *Click Here For Answer*

201 Created

Which of the following status codes indicates that a requested resource was not found? A) 200 B) 404 C) 500 D) 403

▼ *Click Here For Answer*

B) 404