Interview Questions & Answers on REST & RESTful API Design Principles

---

1️⃣ Basic Questions

1. What is REST, and how does it differ from SOAP?

REST (Representational State Transfer) is an architectural style that defines a set of constraints for designing web services. It primarily uses standard HTTP methods and focuses on resources rather than operations.

Differences between REST and SOAP:

- **Protocol:** REST is an architectural style, while SOAP (Simple Object Access Protocol) is a strict protocol.
- **Data Format:** REST typically uses JSON, while SOAP uses XML.
- **Performance:** REST is lightweight and faster than SOAP, which has a higher overhead due to XML formatting and WS-Security.
- **Flexibility:** REST supports multiple data formats (JSON, XML, HTML, etc.), while SOAP is XML-based.
- **Statefulness:** REST is stateless; SOAP can be stateful.

---

2. What are the six constraints of REST architecture?

The six constraints that define REST are:

1. **Client-Server Architecture** – The client and server are independent of each other, allowing for better scalability.
2. **Statelessness** – Each request from the client contains all the necessary information; the server does not store session data.
3. **Cacheability** – Responses can be cached to improve performance and reduce server load.
4. **Layered System** – An API can be designed in layers (e.g., authentication, load balancing, security) without affecting clients.

5. **Uniform Interface** – Standardized communication methods using HTTP verbs (`GET`, `POST`, `PUT`, `DELETE`).

6. **Code on Demand (Optional)** – The server can send executable code (e.g., JavaScript) to the client.

---

3. What is the difference between a REST API and a RESTful API?

- **REST API**: Any API that follows some REST principles but may not strictly adhere to all six constraints.

- **RESTful API**: An API that fully follows all REST constraints and principles.

---

4. What is a resource in REST, and how is it represented?

- A **resource** is any object or entity that can be accessed via the API (e.g., users, orders, products).

- Resources are represented using **URIs** (Uniform Resource Identifiers).

    - Example: `/users/{id}`, `/products/{id}`

---

5. What are endpoints in a REST API?

- An **endpoint** is a specific URL where a client interacts with a resource.

- Example:

    - `GET /users/{id}` – Retrieve user details.

    - `POST /orders` – Create an order.

2️⃣ HTTP Methods & Status Codes

6. Explain the difference between `GET`, `POST`, `PUT`, `PATCH`, and `DELETE`.

- `GET` – Retrieve resource data (safe, idempotent).
- `POST` – Create a new resource (not idempotent).
- `PUT` – Update a resource completely (idempotent).
- `PATCH` – Partially update a resource (not always idempotent).
- `DELETE` – Remove a resource (idempotent).

---

7. When would you use `PUT` vs. `PATCH`?

- **Use `PUT`** when replacing an entire resource (e.g., updating all fields).
- **Use `PATCH`** when making partial updates (e.g., changing only one field).

---

8. What are the commonly used HTTP status codes in REST APIs?

- `200 OK` – Successful request.
- `201 Created` – Resource successfully created.
- `204 No Content` – Successful request, but no response body.
- `400 Bad Request` – Client-side error (e.g., malformed request).
- `401 Unauthorized` – Authentication is required.
- `403 Forbidden` – Client lacks permission.
- `404 Not Found` – Resource does not exist.
- `500 Internal Server Error` – Unexpected server-side issue.

3️⃣ RESTful API Design & Best Practices

9. What are the best practices for designing RESTful APIs?

✅ Use **plural nouns** for resource names (`/users`, not `/user`).
✅ Implement **proper HTTP status codes**.
✅ Support **versioning** (`/v1/resources`).
✅ Use **pagination** for large datasets (`?page=2&limit=20`).
✅ Implement **rate limiting** to prevent abuse.
✅ Use **OAuth or JWT** for authentication.

---

10. How do you design a RESTful API for a blogging platform?

Endpoints could include:

- `GET /posts` – Get all posts
- `POST /posts` – Create a new post
- `GET /posts/{id}` – Get a specific post
- `POST /posts/{id}/comments` – Add a comment

---

11. What is HATEOAS in REST?

HATEOAS (Hypermedia as the Engine of Application State) is a principle where API responses include links to relevant actions.
Example:

json

```
{
  "id": 1,
  "name": "John",
  "links": {
    "self": "/users/1",
    "orders": "/users/1/orders"
  }
}
```

---

12. How do you handle authentication and authorization in REST APIs?

✅ Use **OAuth 2.0** for user authentication.
✅ Use **JWT (JSON Web Token)** for session management.
✅ Implement **API keys** for third-party access.

4️⃣ Advanced & Real-World Questions

13. How does caching work in REST APIs?

- Use HTTP headers:
    - `Cache-Control: max-age=3600` (cache for 1 hour).
    - `ETag` – Versioning for resources.

---

14. How do you implement pagination in REST APIs?

Use query parameters:

- `GET /users?page=2&limit=10`

---

15. How does versioning work in REST APIs?

- URI-based: `/v1/resource`
- Header-based: `Accept-Version: v1`
- Query-based: `?version=1`

---

16. Explain the differences between REST, GraphQL, and gRPC.

Feature	REST	GraphQL	gRPC
Data Format	JSON, XML	JSON	Protocol Buffers

Query Flexibility	Fixed Endpoints	Custom Queries	Strict Methods
Performance	Medium	High	Very High
Use Case	General APIs	Complex Data Fetching	Low-latency Services

---

17. How would you improve the performance of a REST API?

✅ Enable **caching**.
✅ Use **gzip compression**.
✅ Implement **asynchronous processing**.
✅ Use **database indexing**.

---

18. Can a REST API be stateful?

No, REST APIs should be **stateless**. However, some services may use **session-based authentication**, making them stateful.

---

19. How does REST handle security vulnerabilities?

- Prevent **SQL injection** by using parameterized queries.
- Use **CSRF tokens** to prevent cross-site request forgery.
- Implement **HTTPS** for secure communication.

5️⃣ Summary

- REST follows six constraints: **stateless, cacheable, client-server, uniform interface, layered system, code-on-demand (optional)**.
- Use **proper HTTP methods** and **status codes**.
- **Security** (OAuth, JWT) and **performance** (caching, pagination) are crucial.