Basic

1. What is a REST API?

uses secure http methods for secure data transformation.

- REST (Representational State Transfer) API provides a way for applications to interact with each other and exchange data using standard web protocols. It's commonly used in modern web development for building scalable and efficient applications that can communicate and share information seamlessly.

2. What are the main HTTP methods used in RESTful services?

- The main HTTP methods used in RESTful services are:
  - **GET**: Retrieve a resource.
  - **POST**: Create a new resource.
  - **PUT**: Update an existing resource.
  - **DELETE**: Remove a resource.
  - **PATCH**: Partially update a resource.
  - **OPTIONS**: Get information about the communication options available for a resource.

3. Explain the concept of RESTful URI.

- RESTful URIs (Uniform Resource Identifiers) are URLs that uniquely identify resources. They follow a hierarchical pattern and use nouns (resources) rather than verbs (actions) to represent entities and their relationships. For example, `/users/123` identifies a specific user with ID 123.

Intermediate

1. What are the best practices for designing a REST API?

- Best practices include:
  - Using nouns (resources) instead of verbs in URIs.
  - Using HTTP methods correctly and consistently.
  - Providing resource representations in multiple formats (like JSON, XML).
  - Supporting query parameters for filtering, sorting, and pagination.
  - Versioning APIs to manage changes gracefully.
  - Documenting APIs comprehensively with clear descriptions and examples.

2. How do you handle authentication in a REST API?

- Authentication in REST APIs is typically handled using tokens (like JWT) or API keys. Common methods include:
  - **Token-based authentication**: Using JWT for stateless authentication.
  - **OAuth**: For delegated authorization, allowing third-party applications limited access.
  - **Basic Authentication**: Sending username and password encoded in Base64 (secured over HTTPS).
  - **API Keys**: Unique keys sent with each request to authenticate clients.

3. What is the purpose of status codes in a REST API?

- Status codes convey the outcome of an HTTP request. They indicate whether a request was successful, encountered an error, or requires further action. Some common status codes:
  - **2xx**: Successful (e.g., 200 OK, 201 Created).
  - **3xx**: Redirection (e.g., 301 Moved Permanently).
  - **4xx**: Client error (e.g., 400 Bad Request, 404 Not Found).
  - **5xx**: Server error (e.g., 500 Internal Server Error).

Advanced

1. Explain the concept of HATEOAS in RESTful services.

- HATEOAS (Hypermedia as the Engine of Application State) allows a REST client to interact with a server dynamically through hypermedia links provided in the responses. It enables discovering available actions and transitioning between application states without relying on prior knowledge of server APIs.

2. How do you implement rate limiting in a REST API?

- Rate limiting restricts the number of API requests a client can make within a specific time frame. Implementation involves:
  - Setting limits based on IP address, API key, or user.
  - Using tokens or headers to track request counts.
  - Throttling requests by returning 429 Too Many Requests status code when limits are exceeded.
  - Providing headers like `X-RateLimit-Limit`, `X-RateLimit-Remaining`, and `X-RateLimit-Reset` to communicate rate limit details.

3. Discuss the pros and cons of using REST vs GraphQL.

- **REST Pros:**
  - Simple and well-established.
  - Flexible with support for different data formats (JSON, XML).
  - Good for hierarchical data and CRUD operations.
- **REST Cons:**
  - Over-fetching or under-fetching of data (when APIs don't match client needs).
  - Multiple endpoints for different data requirements.
  - Limited flexibility in data retrieval.
- **GraphQL Pros:** GraphQL is a query language for APIs. It allows clients to request exactly the data they need, nothing more and nothing less, making it efficient and flexible.
  - Allows clients to request exactly the data they need.
  - Single endpoint for querying and mutating data.
  - Efficient for complex data requirements and reducing network overhead.
- **GraphQL Cons:**

- Requires a learning curve for both clients and servers.
- Increased complexity in server-side implementation.
- Potential security concerns with overly permissive queries.