Principles of REST

- Resource-Based
    - Resources are the fundamental elements in REST API design, identified by URIs.
    - Each resource is represented in a format (JSON, XML) that can be easily consumed by the client.
- Stateless Communication
    - Every API request operates independently, without requiring server-side session information.
    - This design principle aids in scaling and simplifies the server architecture by avoiding session data management.
- Client-Server Separation
    - Promotes separation of concerns: the client handles the user interface, while the server manages data storage and processing.
    - Enhances flexibility, allowing independent development and deployment of client-side and server-side components.
- Uniform Interface
    - Ensures a standardized way of interacting with the resources through HTTP methods.
    - This uniformity simplifies and decouples the architecture, enabling separate evolution of components.
- Layered System
    - Supports a hierarchical architecture with multiple layers, enhancing security and scalability.
    - Clients may interact with an intermediary rather than directly with the end server, transparently improving load balancing and shared caches.
- Cacheability
    - Responses must explicitly indicate their cacheability, reducing the need for some client-server interactions.
    - Effective caching strategies can significantly enhance performance and reduce server load.

Components of REST API

- Resources
    - Central to REST API; anything that can be named, described, and manipulated.
    - Resources are manipulated via standard HTTP methods, making them universally accessible and modifiable.
- HTTP Methods
    - GET: Retrieves data; should be idempotent (safe) and cacheable.
    - POST: Creates a new resource; not idempotent.
    - PUT: Updates an existing resource entirely; idempotent.
    - DELETE: Removes a resource; idempotent.
    - PATCH: Applies partial modifications to a resource; idempotent in practice.
- Representations
    - The form of the resource returned to the client (JSON, XML, HTML), including metadata to guide the client on how to process the resource.
- Stateless Interactions
    - Enhances scalability and reliability by ensuring each request is self-contained.

Additional Considerations

- Security Considerations
    - Despite its stateless nature, REST APIs must be secured with HTTPS, token-based authentication (OAuth, JWT), and proper validation to protect sensitive data.
- Versioning
    - APIs evolve; versioning allows the API to change without breaking existing clients.
    - Common strategies include URI versioning (e.g., /v1/resource), parameter versioning, and header versioning.
- Error Handling
    - REST APIs should return meaningful HTTP status codes along with error messages in the body to help clients understand what went wrong.
    - Common codes include 404 (Not Found), 400 (Bad Request), and 500 (Internal Server Error).
- Rate Limiting
    - To ensure API reliability and availability, rate limiting restricts the number of requests a user can make in a given timeframe.
    - Typically implemented with HTTP headers indicating the current limit, count, and window reset time.
- Documentation
    - Comprehensive documentation is crucial for developer adoption and effective use of the API.
    - Tools like Swagger (OpenAPI) and RAML help in documenting APIs interactively, making it easier for developers to understand and test the API.