#_ Common Backend Interview Questions

1. Difference between PUT, POST, and PATCH:

- **PUT:** It **is** used **to** update **or** create a resource. **If** the resource doesn't exist, a new one will be created. It's idempotent, meaning multiple identical requests should have the same effect as a single request.
- **POST:** Utilized **to** submit data **to** a resource **for** processing. It's non-idempotent and may cause different results with multiple identical requests.
- **PATCH:** Updates **partial** resource data. Unlike PUT, PATCH applies a **partial** update **to** the resource.

2. What is a Payload in a REST API?

A payload refers **to** the data sent with a request **or** response. **In** a REST API, request payload can be **in** the form of JSON, XML, **or** other formats. The payload contains the information necessary **for** the server **or** client **to** process the message.

3. What is a REST Message?

REST messages consist **of** requests **from** clients **to** servers **and** responses **from** servers **to** clients. A REST message often contains an HTTP **method**, **headers**, **URI**, **and payload**.

4. Core Components of an HTTP Request:

- **Method**: The HTTP **method such as GET**, **POST**, **PUT**, **DELETE**.
- **URI**: The Uniform Resource Identifier identifying the resource.
- **Headers**: Metadata associated **with** the request.
- **Body**: The data being sent **with** the request (**if** applicable).

5. Core Components of an HTTP Response:

- **Status** Line: Contains **the** protocol version, status code, **and** status description.
- **Headers**: Metadata associated **with the** response.
- **Body**: The data being returned from **the** server.

6. What is an Idempotent Method and Why is it Important?

Idempotent methods are those that have no additional effect if called more than once with the same input parameters. They are crucial for reliability and consistency, particularly in network environments where failures are commonplace.

7. Difference Between Idempotent and Safe HTTP Methods:

- **Idempotent Methods**: These methods (GET, PUT, DELETE, HEAD, OPTIONS) produce the same result regardless of the number of times they are called.
- **Safe Methods**: These methods (GET, HEAD, OPTIONS) do not modify resources.

8. Explain Caching in a RESTful Architecture:

Caching involves storing copies of files in strategic locations to improve performance and efficiency. In REST, responses can be explicitly marked as cacheable or non-cacheable. Caching mechanisms could be implemented at various levels like browser, proxy, or server.

9. How Do You Handle Concurrent Modifications?

Concurrent modifications can be handled using optimistic locking mechanisms, ETags, or timestamps. When a resource is read, its version is also retrieved. Upon update, if the version has changed, it indicates a concurrent modification.

10. Explain the OAuth 2.0 Authorization Framework:

OAuth 2.0 enables third-party applications to obtain limited access to an HTTP service. It works by delegating user authentication to the service that hosts the user's account and authorizing third-party applications to access the user account.

11. How Would You Design a Rate-Limiter?

A Rate-Limiter can be designed using various algorithms like Token Bucket, Leaky Bucket, or using a sliding log mechanism. It controls the rate of events by delaying actions that comply with the given rate limits.

12. How Can You Secure RESTful Web Services?

- **Authentication**: Verify the identity of the requesting user or system.
- **Authorization**: Check permissions of authenticated users.
- **Encryption**: Encrypt data transmitted over the network using protocols like HTTPS.
- **Validation**: Validate input data to protect against injection attacks.

13. Explain the Role of Middleware in Backend Development:

Middleware functions are those that have access to the request object, response object, and the next function in the application's request-response cycle. They can execute any code, modify the request and response objects, end the request-response cycle, or call the next function in the stack.

14. What are the Principles of a Twelve-Factor App?

The Twelve-Factor App methodology is a set of best practices to build modern web applications, or software-as-a-service apps. The principles include codebase, dependencies, config, backing services, build-release-run, processes, port binding, concurrency, disposability, dev/prod parity, logs, and admin processes.

15. Explain Database Sharding and its Advantages:

Database sharding involves breaking a large database into smaller, more manageable pieces or "shards". Each shard holds a subset of the data and operates independently of the others. Advantages include improved performance, easier scalability, and enhanced management capabilities.

16. How Do You Handle Long-Running Transactions?

For long-running transactions, it's important to consider:

- Breaking the transaction into smaller, manageable pieces.
- Using asynchronous processing mechanisms.
- Implementing appropriate timeout and retry logic.
- Applying compensation transactions for rollback mechanisms if necessary.

17. How Can Microservices Communication be Secured?

Microservices communication can be secured through:

- Mutual TLS (mTLS) for encrypted communication and identity verification.

- API Gateways to enforce security policies.
- JSON Web Tokens (JWT) or OAuth 2.0 for authentication and authorization.
- Network policies and segmentation to control traffic between services.

18. Describe the CAP Theorem:

The CAP theorem states that it's impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees: Consistency, Availability, and Partition tolerance.

19. How Do You Design a System to Handle 10 Million Requests Per Second?

This involves multiple considerations:

- Load balancing to distribute incoming network traffic across multiple servers.
- Data partitioning and sharding for effective data management and retrieval.
- Implementing caching mechanisms to reduce database load.
- Utilizing Content Delivery Networks (CDN) to serve static assets.
- Employing auto-scaling to handle traffic spikes.

20. Explain the Differences Between Monolithic and Microservices Architecture:

Monolithic architecture is a traditional model where all application code is within a single codebase. Microservices architecture breaks down a traditional application into smaller, self-contained services, which can be developed, deployed, and scaled independently.

21. Discuss Strategies for Service Discovery in a Microservices Architecture:

Service Discovery is essential in microservices architectures for locating services at runtime. Strategies include using a DNS, a service registry like Eureka, or a service mesh like Istio or Linkerd.

22. What are the Different Types of Database Indexes and How Do They Work?

Indexes are database structures that improve query speed. Types include:

- Single-level index: A simple index with a reference to the data.

- Multi-level index: An index on another index.
- Clustered index: The physical ordering of data storage is rearranged.
- Non-clustered index: Logical ordering does not match physical ordering.

23. How Do You Implement Authentication in a Microservices Architecture?

Authentication can be implemented using:

- **JWT (JSON Web Tokens):** Stateless, and can be verified by microservices without needing to check a central authority every time.
- **OAuth 2.0:** Delegated authorization mechanism, useful for providing third-party apps access.
- **Single Sign-On (SSO):** Allows users to authenticate once and gain access to different services.

24. What Strategies Would You Use for Efficient Logging and Monitoring in Distributed Systems?

Effective logging and monitoring in distributed systems can be achieved through:

- Centralized logging using tools like ELK stack or Graylog.
- Implementing tracing using OpenTracing or OpenTelemetry.
- Utilizing monitoring solutions like Prometheus, Grafana, or New Relic.
- Ensuring logs are structured and include necessary context for debugging.

25. Best Practices in Developing a RESTful Web Service:

- Use nouns to represent resources and verbs for actions.
- Implement stateless operations.
- Leverage standard HTTP methods.
- Utilize status codes correctly.
- Version your API.
- Handle errors gracefully and provide helpful error messages.
- Use OAuth for security.