Interview - API

Interview Questions

(Practice Project)

PW SKILLS

Easy

1. What are the key differences between SOAP and REST?

Ans:
- SOAP (Simple Object Access Protocol) is a protocol, while REST (Representational State Transfer) is an architectural style.
- REST supports multiple data formats (JSON, XML, etc.), while SOAP uses only XML.
- REST is generally faster and more lightweight than SOAP.
- SOAP has built-in security and reliability features, while REST inherits security from the underlying protocol.
- REST uses standard HTTP methods, while SOAP defines its own standards.
- REST is more commonly used for web services and public APIs, while SOAP is often used in enterprise environments.

2. What are the best practices for designing URIs in RESTful web services?

Ans:
- Use plural nouns for resources (e.g., /users instead of /user)
- Use hyphens or underscores for long names (e.g., /authorized-users)
- Use lowercase letters in URIs
- Maintain backward compatibility when updating URIs
- Use appropriate HTTP methods (GET, POST, PUT, DELETE) without including them in the URI
- Use forward slashes to indicate hierarchy (e.g., /users/{id}/address)
- Keep URIs simple, readable, and self-explanatory

3. What are the best practices for developing RESTful web services?

Ans:
- Use JSON as the primary data format for requests and responses
- Set appropriate Content-Type headers
- Implement proper error handling with appropriate HTTP status codes
- Include pagination and filtering for large data sets
- Implement secure communication (SSL/TLS)
- Use role-based access control for security
- Implement caching when appropriate to improve performance
- Version your APIs to maintain compatibility
- Use plural nouns for resource endpoints
- Provide comprehensive API documentation

4. What are idempotent methods in REST, and why are they important?

Ans: Idempotent methods are those that produce the same result regardless of how many times they're called. In REST:
- GET, PUT, DELETE, HEAD, OPTIONS, and TRACE are idempotent
- POST is not idempotent

Idempotency is important because it allows clients to safely retry requests without unintended side effects, improving reliability and fault tolerance in distributed systems.

5. What are the main differences between REST and AJAX?

Ans:
- REST is an architectural style for designing networked applications, while AJAX is a technique for creating asynchronous web applications.
- REST focuses on resource-based interactions, while AJAX is about updating parts of a web page without reloading the entire page.
- REST is typically used for server-side API design, while AJAX is a client-side technique.
- REST can use various data formats, while AJAX traditionally uses XML (though JSON is now common).
- REST is stateless, while AJAX can maintain state in the browser.

6. What are the core components of an HTTP Request?

Ans: An HTTP Request consists of:
- Method/Verb (GET, POST, PUT, DELETE, etc.)
- URI (Uniform Resource Identifier)
- HTTP Version
- Request Headers
- Request Body (optional, depending on the method)

7. What are the core components of an HTTP Response?

Ans: An HTTP Response consists of:
- Response Status Code (e.g., 200 OK, 404 Not Found)
- HTTP Version
- Response Headers
- Response Body

8. What are the main differences between PUT and POST methods in REST?

Ans:
- PUT is typically used for updating existing resources or creating new ones at a specific URI. It's idempotent.
- POST is used for creating new resources where the server decides the URI. It's not idempotent.
- PUT requests are usually sent to a specific resource URI, while POST requests are sent to a general resource collection URI.
- Multiple identical PUT requests will have the same effect as a single request, while multiple POST requests may create multiple resources.

9. How does REST architecture contribute to scalability?

Ans: REST's statelessness is key to its scalability:
- Each request contains all the information needed to process it, allowing for easier load balancing.
- No need to maintain session state on the server, simplifying horizontal scaling.
- Caching can be more effectively implemented, reducing server load.
- The uniform interface principle simplifies the overall system architecture.

10. How do you choose between SOAP and REST for a web service?

Ans: Consider the following factors:
- Data format requirements (REST supports multiple formats, SOAP uses XML)
- Need for strict contracts (SOAP provides WSDLs)
- Security requirements (SOAP has built-in security features)
- Performance and scalability needs (REST is generally more lightweight)
- Support for transactions (SOAP has better support for complex transactions)
- Client types and ease of consumption (REST is often easier for web and mobile clients)

Medium

11. What are the key differences between REST and WebSocket APIs?

Ans:
- **Communication model:** REST follows request-response, WebSocket allows full-duplex communication.
- **State:** REST is stateless, WebSockets are stateful.
- **Protocol:** REST typically uses HTTP/HTTPS, WebSocket has its own protocol.
- **Real-time capabilities:** WebSockets are better suited for real-time applications.
- **Connection:** REST opens a new connection for each request, WebSocket maintains a persistent connection.
- **Use cases:** REST for general API needs, WebSocket for real-time updates and notifications.

12. How can you implement security in RESTful web services?

Ans:
- Use HTTPS (TLS/SSL) for encrypted communication
- Implement authentication (e.g., JWT, OAuth)
- Use authorization mechanisms (role-based access control)
- Implement input validation to prevent injection attacks
- Use API keys for identifying and tracking API usage
- Implement rate limiting to prevent abuse
- Keep sensitive data out of URLs
- Use secure headers (e.g., CORS, Content-Security-Policy)

13. What tools can be used to test RESTful Web Services?

Ans: Popular tools for testing REST APIs include:
- **Postman:** For sending requests and analyzing responses
- **Swagger/OpenAPI:** For API documentation and testing
- **JMeter:** For performance and load testing
- **REST Assured:** For automated API testing in Java
- **cURL:** Command-line tool for making HTTP requests
- **SoapUI:** Despite the name, it's also useful for testing REST APIs
- **Insomnia:** A lightweight alternative to Postman

14. Is there a limit to the payload size in POST methods?

Ans: Theoretically, there's no standard limit to the payload size for POST requests. However:
- Practical limits may be imposed by servers, frameworks, or network configurations.
- Large payloads can impact performance and resource usage.
- It's generally recommended to keep payloads reasonably sized and consider alternatives (like file uploads) for very large data transfers.

15. How does HTTP Basic Authentication work?

Ans: HTTP Basic Authentication works as follows:
- The client sends a request to a protected resource.
- The server responds with a 401 Unauthorized status and a WWW-Authenticate header.
- The client prompts the user for credentials.
- The client concatenates the username and password with a colon (username:password).
- This string is base64 encoded.
- The client sends another request with an Authorization header containing "Basic" followed by the encoded string.
- The server decodes and verifies the credentials, granting access if valid.

Note: Basic Authentication should only be used over HTTPS as the credentials are essentially sent in plain text.

16. What's the difference between idempotent and safe HTTP methods?

Ans:
- **Safe methods:** These do not modify resources on the server. They are safe to call without risk of data modification or corruption. Examples: GET, HEAD, OPTIONS.
- **Idempotent methods:** These produce the same result no matter how many times they are called. They may modify resources, but repeated calls won't cause additional changes. Examples: GET, HEAD, PUT, DELETE, OPTIONS, TRACE.

Key differences:
- All safe methods are idempotent, but not all idempotent methods are safe.
- Safe methods are read-only, while some idempotent methods can modify data.
- POST is neither safe nor idempotent.

17. How can you handle versioning in RESTful APIs?

Ans: There are several approaches to API versioning:
- URI Path Versioning: Include the version in the URI (e.g., /api/v1/users)
- Query Parameter Versioning: Use a query parameter (e.g., /api/users?version=1)
- Custom Header Versioning: Use a custom header (e.g., X-API-Version: 1)
- Accept Header Versioning: Use the Accept header (e.g., Accept: application/vnd.company.api.v1+json)

Each approach has pros and cons regarding simplicity, client compatibility, and caching. URI path versioning is often preferred for its simplicity and explicitness.

Hard

19. What are the key principles of REST, and why is statelessness important?

Ans: REST (Representational State Transfer) is an architectural style for building scalable web services. The key principles of REST include statelessness, client-server separation, a uniform interface, cacheability, and layered system architecture. Statelessness is crucial because it ensures that each request from a client contains all the information necessary to process the request. This simplifies server design and improves scalability by not requiring the server to maintain the state of each client session.

20. What is the difference between POST and PUT HTTP methods in RESTful APIs?

Ans: Both POST and PUT are used to send data to the server in RESTful APIs, but they have different purposes. The POST method is used to create a new resource. Each POST request usually results in the creation of a unique resource. The PUT method, on the other hand, is used to update an existing resource or create it if it does not exist. PUT is idempotent, meaning multiple identical requests should produce the same result.

21. How does FastAPI automatically generate API documentation, and what are the benefits?

Ans: FastAPI automatically generates interactive API documentation using Swagger UI and ReDoc. This is possible because FastAPI is built on top of Python's type hints and Pydantic models, which provide a clear structure of the data expected by the API. The benefits include easier API testing, better developer experience, and quicker integration for users of the API.

22. What is the role of the `Depends` function in FastAPI, and how does it support dependency injection?

Ans: The `Depends` function in FastAPI allows you to define dependencies for your endpoint functions. Dependency injection is a design pattern that enables the creation of objects outside of a class and makes those objects available to the class. In FastAPI, this makes the code modular, reusable, and easier to test. It allows for clean separation of concerns by injecting dependencies like database connections, authentication checks, or other reusable functions.

23. What is JWT, and how does its structure ensure security and integrity?

Ans: JWT (JSON Web Token) is a compact, self-contained token used for securely transmitting information between parties as a JSON object. It consists of three parts: the header, the payload, and the signature. The header contains the type of token and the signing algorithm. The payload contains claims, which are statements about an entity (typically the user) and additional data. The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

24. How does the PATCH HTTP method differ from PUT, and when would you use it?

Ans: The PATCH method is used for partially updating a resource, unlike PUT, which replaces the entire resource with the new data. PATCH is used when only a small portion of the resource needs to be updated, thus saving bandwidth and reducing the risk of overwriting unintended parts of the resource. This method is particularly useful in scenarios where the resource is large, and only a minor change is required.

25. Why is API versioning important, and what are some common strategies to implement it?

Ans: API versioning is essential for maintaining backward compatibility as an API evolves. It allows developers to introduce new features or changes without breaking existing clients. Common strategies for API versioning include URL versioning (e.g., `/v1/resource`), query parameter versioning (e.g., `/resource?version=1`), and header versioning (e.g., specifying the version in the HTTP headers). Each strategy has its pros and cons, depending on the specific use case.

26. What are the benefits of using OAuth2 for authorization, and how does the authorization code grant flow work?

Ans: OAuth2 is an open standard for authorization that provides secure, delegated access to resources without exposing user credentials. The authorization code grant flow, which is commonly used for web applications, involves several steps: the client application redirects the user to the authorization server, the user approves the request, the authorization server sends an authorization code back to the client, and the client exchanges this code for an access token. This flow is secure because the access token is never exposed to the user's browser.

27. What are the typical HTTP status codes returned by a RESTful API, and what do they signify?

Ans: HTTP status codes are standardized codes returned by the server to indicate the outcome of an HTTP request. Common codes include:

- 200 OK: The request was successful.
- 201 Created: A resource was successfully created.
- 400 Bad Request: The server could not understand the request due to invalid syntax.
- 401 Unauthorized: Authentication is required and has failed or not been provided.
- 404 Not Found: The requested resource could not be found.
- 500 Internal Server Error: The server encountered an unexpected condition that prevented it from fulfilling the request.