Rest API

1. Rest API?

A REST API provides endpoints that allow clients to communicate with a server over the internet. Each endpoint represents a specific resource or action, such as retrieving data, creating new data, updating existing data, or deleting data. Clients send requests to these endpoints using HTTP methods like GET, POST, PUT, PATCH, and DELETE, and the server responds with the requested data or performs the requested action.

This client-server communication follows the principles of REST (Representational State Transfer), which emphasizes statelessness, a uniform interface, and resource-based interactions.

So, in total, REST APIs support these **seven HTTP methods**:

1. **GET** – Retrieve data (with response body).
2. **POST** – Create a new resource.
3. **PUT** – Update/replace a resource.
4. **PATCH** – Partially update a resource.
5. **DELETE** – Remove a resource.
6. **OPTIONS** – Fetch allowed HTTP methods for a resource.
7. **HEAD** – Retrieve only headers (without response body).

2. What are the key characteristics of REST APIs?

Stateless: Each request from a client must contain all the necessary information, as the server does not store session data.
Client-Server Architecture: The client and server are independent.
Cacheable: Responses can be cached to improve performance.
Layered System: APIs can have multiple layers for security, load balancing, etc.
Uniform Interface: Uses standard HTTP methods and URIs for interaction.

3. What is the difference between REST and SOAP?

Feature	REST	SOAP
Protocol	Uses HTTP	Uses HTTP, SMTP, TCP
Format	JSON, XML, etc.	Only XML
Performance	Faster & lightweight	Slower due to XML overhead
Stateless	Yes	Can be stateful or stateless
Security	Uses OAuth, JWT, etc.	Uses WS-Security

4. What is Idempotency in REST APIs?

An idempotent operation means that making multiple identical requests will not change the server state beyond the first request.
Idempotent methods: GET, PUT, DELETE, HEAD, OPTIONS
Non-idempotent methods: POST, PATCH
Example: Calling DELETE /users/123 multiple times should delete the user once and return the same response.

5. What is the difference between PUT and PATCH?

PUT – Replaces the entire resource with a new one.
PATCH – Partially updates a resource (only specified fields).
Example:

- PUT /users/1 → Replace the entire user object.
- PATCH /users/1 → Update only specific fields like email or name.

6. What are common HTTP status codes in REST APIs?

200 OK – Success
201 Created – Resource successfully created
204 No Content – Success but no response body
400 Bad Request – Client error (invalid request)
401 Unauthorized – Authentication required
403 Forbidden – Client does not have access
404 Not Found – Resource does not exist
500 Internal Server Error – Generic server error

7. How do you secure a REST API?

Authentication: Use OAuth 2.0, JWT, or API keys
Authorization: Implement role-based access control (RBAC)
CORS (Cross-Origin Resource Sharing): Restrict API access to trusted origins
Rate Limiting & Throttling: Prevent abuse by limiting API calls
Input Validation & Sanitization: Prevent SQL Injection, XSS attacks
HTTPS: Encrypt data in transit

8. What is API rate limiting?

API rate limiting restricts the number of API requests a user or client can make in a specific time period.
Example: 1000 requests per hour per user.
Used to prevent abuse and ensure fair usage.

9. What are the key annotations used to create a REST API in Spring Boot?

Spring Boot provides the following annotations for building REST APIs:

@RestController – Marks a class as a RESTful controller (combines @Controller and @ResponseBody).
@RequestMapping – Defines the base URL for the API.
@GetMapping, @PostMapping, @PutMapping, @DeleteMapping – Maps HTTP methods to controller methods.
@PathVariable – Extracts values from the URI path.
@RequestParam – Extracts query parameters from the request.
@RequestBody – Maps request JSON data to a Java object.
@ResponseEntity – Used to customize HTTP responses.

10. How does Spring Boot support the creation of RESTful APIs?

Spring Boot provides excellent support for creating RESTful APIs using Spring MVC and embedded HTTP servers like Tomcat, Jetty, or Undertow. Developers can define RESTful endpoints using `@RestController` and `@RequestMapping` annotations, handle request and response payloads with ease, and leverage features like content negotiation, validation, and error handling.

11. What is the difference between @RestController and @Controller?

@RestController = @Controller + @ResponseBody
@Controller is used for rendering views (HTML, JSP), whereas @RestController is used for REST APIs that return JSON or XML.

```java
@RestController
@RequestMapping("/users")
public class UserController {

    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        return new User(id, "John Doe");
    }
}
```

The above returns a JSON response. If we used @Controller, it would return a view name instead.

12. How do you handle exceptions in a Spring Boot REST API?

Exception handling can be done using:
@ExceptionHandler – Handles specific exceptions in a controller.
@ControllerAdvice – Global exception handling across multiple controllers.
ResponseEntityExceptionHandler – Provides default exception handling.

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<String> handleUserNotFound(UserNotFoundException ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }
}
```

If UserNotFoundException occurs, it returns **404 Not Found** with a custom message.

13. What are the different types of API authentication methods?

Basic Authentication – Uses username: password (Base64 encoded).
Bearer Token Authentication – Uses a token in the Authorization header.
OAuth 2.0 – Secure authorization framework for third-party access.
JWT (JSON Web Token) – Used for stateless authentication.
API Keys – A unique key provided to authenticate requests.

14. How do you implement pagination in a Spring Boot REST API?

Pagination is implemented using Spring Data JPA's Pageable interface.

```
@GetMapping("/users")
public Page<User> getUsers(Pageable pageable) {
    return userRepository.findAll(pageable);
}
```

Example API call: /users?page=0&size=10&sort=name,asc

15. What is an API Gateway?

An API Gateway is a management tool that acts as an entry point for client requests, providing functionalities like authentication, rate limiting, logging, and request routing.
Examples:
AWS API Gateway, Apigee
Kong, Zuul

16. How do you secure a Spring Boot REST API?

JWT (JSON Web Token) – Token-based authentication.
OAuth 2.0 – Secure access delegation.
Spring Security – Manages authentication & authorization.
API Keys – Secure APIs using a unique key.
HTTPS – Encrypt data in transit.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/public").permitAll()
            .antMatchers("/admin").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .httpBasic();
    }
}
```

This restricts access to the /admin endpoint.

17. What is an API Gateway in Spring Boot?

An API Gateway acts as an entry point for all client requests, handling authentication, load balancing, and routing.
Spring Cloud Gateway is the most used gateway in Spring Boot microservices.

```
spring:
  cloud:
    gateway:
      routes:
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/users/**
```

This routes /users/** requests to the USER-SERVICE.

18. Why are we using REST services? Why is popular?

Simplicity: RESTful services are based on a simple architectural style that uses standard HTTP methods (GET, POST, PUT, DELETE) and resource-based URLs. This simplicity makes it easy to understand and use RESTful APIs.

Scalability: RESTful services are stateless, meaning each request from a client contains all the information needed by the server to fulfil the request. This statelessness makes it easier to scale RESTful services horizontally by adding more servers to handle increased load.

Flexibility: RESTful services support multiple data formats, including JSON, XML, and plain text, making them suitable for a wide range of clients and applications.

Interoperability: RESTful services use standard HTTP protocols, making them compatible with a variety of platforms, programming languages, and devices. This interoperability enables communication between different systems and components.

Caching: RESTful services leverage the built-in caching mechanisms of HTTP to improve performance and reduce server load. Clients can cache responses from the server to avoid unnecessary round trips for the same data.

Statelessness: RESTful services are stateless, meaning each request from a client contains all the information needed by the server to fulfil the request. This simplifies server-side logic and improves reliability.

Uniform Interface: RESTful services follow a uniform interface, making it easier to develop, understand, and maintain APIs. This uniformity includes standard methods (GET, POST, PUT, DELETE), resource-based URLs, and self-descriptive messages.

Overall, RESTful services provide a lightweight, scalable, and interoperable way to build distributed systems and expose APIs for communication between different software components. They have become the preferred choice for building web services and APIs due to their simplicity, flexibility, and compatibility with modern web development practices.

19. What is HATEOAS in REST APIs?

HATEOAS (Hypermedia as the Engine of Application State) is a REST principle that provides links to related resources in the API response, allowing clients to navigate dynamically.

```
{
  "userId": 1,
  "name": "John Doe",
  "links": [
    { "rel": "self", "href": "/users/1" },
    { "rel": "orders", "href": "/users/1/orders" }
  ]
}
```

20. How do you document a Spring Boot REST API?

API documentation is done using:
Swagger (Springdoc OpenAPI)

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.0.0</version>
</dependency>
```

- Access Swagger UI at /swagger-ui.html.

Postman – API testing & documentation.
Spring REST Docs – Generates documentation from tests.

----------END----------