[***https://spring.io/guides/gs/rest-service***](https://spring.io/guides/gs/rest-service)

# 🔥 REST API Interview Questions for 7+ Years Experience

# ✅ 1. How do you design scalable REST APIs in a microservices architecture?

* **Key topics to touch:**
  + API Gateway (e.g., Spring Cloud Gateway)
  + Centralized logging and rate limiting
  + Service discovery (e.g., Eureka)
  + Versioning strategies
  + Load balancing and horizontal scaling

# ✅ 2. How do you handle schema evolution in REST APIs?

* **Discuss:**
  + API versioning via URI or headers
  + Avoiding breaking changes (backward compatibility)
  + Adding fields (safe), not removing or renaming

# ✅ 3. How do you handle authentication and authorization in REST APIs?

* **Real-time response:**
  + Use JWT tokens for stateless auth
  + OAuth2 for delegated access
  + Role-based access with Spring Security
  + Refresh token mechanism
  + Token expiry and validation filters

# ✅ 4. How do you ensure your REST API is fault-tolerant and resilient?

* **Answer with tools and patterns:**
  + Retry, Fallback, Timeout, Circuit Breaker using Resilience4j
  + Use bulkheads for service isolation
  + Async communication via messaging (Kafka, RabbitMQ)

# ✅ 5. How do you test REST APIs in real-time projects?

* **Tools and practices:**
  + Postman for manual testing
  + RestAssured or MockMvc for automated tests
  + Use contract testing (e.g., Pact)
  + Swagger-based validation

# ✅ 6. How do you handle API documentation and maintain it?

* **Use:**
  + springdoc-openapi, swagger-ui
  + Version control for OpenAPI specs
  + Auto-generate documentation from annotations
  + Developer portal integration

# ✅ 7. How do you enforce input validation in REST APIs?

* **Use:**
  + @Valid, @Validated, @NotNull, @Size, etc.
  + Global exception handling with @ControllerAdvice
  + Customize error messages and response formats

# ✅ 8. How do you monitor and log REST APIs in production?

* **Use:**
  + Centralized logging (ELK, Graylog)
  + Metrics via Micrometer + Prometheus + Grafana
  + Tracing with Zipkin/Jaeger
  + Log correlation with request IDs

# ✅ 9. How do you handle rate limiting and throttling in APIs?

* **Answer:**
  + At API Gateway level using Redis/Bucket4j
  + Use Spring filters to reject high-frequency calls
  + HTTP 429 status for throttling

# ✅ 10. Explain a situation where a REST API caused production issues and how you handled it.

* **Share a real-world experience:**
  + Debugging steps (logs, trace, metrics)
  + Rollback or hotfix process
  + Root cause analysis and permanent fix
  + Preventive measures added later (tests, alerts)

# ✅ 11. How do you deal with backward compatibility in REST APIs?

* **Strategies:**
  + API versioning
  + Add fields instead of removing
  + Use default values for new fields
  + Deprecated annotations with documentation

# ✅ 12. What tools do you use for API performance testing and profiling?

* **Tools:**
  + JMeter, Gatling, k6
  + JVM profilers (VisualVM, YourKit)
  + Spring Actuator + Micrometer metrics

# ✅ 13. How do you handle partial updates in REST? PATCH vs PUT?

* PATCH = partial update, PUT = full replacement
* Use PATCH when only updating few fields
* Handle field nullability and merging logic carefully

# ✅ 14. What is HATEOAS and do you use it in real projects?

* HATEOAS = discoverability via links
* Rarely used in external APIs
* More common in HAL/Hypermedia-driven APIs

# ✅ 15. How do you secure sensitive data in your REST APIs?

* Data encryption at rest and in transit
* Mask sensitive data in logs
* Avoid exposing sensitive info (like internal IDs)
* Validate all incoming data

# ✅ 16. How do you handle distributed transactions across REST microservices?

* Concepts:
  + Saga pattern (orchestration/choreography)
  + Eventual consistency
  + Compensation-based rollback

# ✅ 17. How do you deal with versioning in REST APIs?

* Approaches:
  + URI-based: /api/v1/resource
  + Header-based: Accept: application/vnd.company.v1+json
  + Parameter-based: /api/resource?version=1

# ✅ 18. What are the best practices for designing resource URIs?

* RESTful URI design tips:
  + Use nouns, not verbs → /users not /getUsers
  + Use plural → /orders, not /order
  + Nesting only when logically necessary → /users/{id}/orders
  + Avoid deep nesting

# ✅ 19. How do you ensure REST APIs are cacheable?

* Techniques:
  + Use HTTP headers: Cache-Control, ETag, If-Modified-Since
  + Use CDN/API Gateway caching
  + Separate cacheable (GET) and non-cacheable methods (POST/PUT)

# ✅ 20. How do you implement role-based access control (RBAC) in REST APIs?

* Tools:
  + Spring Security with annotations: @PreAuthorize("hasRole('ADMIN')")
  + Role-permission mapping
  + Token-based authorization with JWT

# ✅ 21. How do you deal with inconsistent response formats from legacy systems or third-party APIs?

* Solutions:
  + Create adapters or transformers in service layer
  + Map external models to internal DTOs
  + Handle exceptions/failures gracefully with fallback data

# ✅ 22. What’s your strategy for deploying and evolving REST APIs in a CI/CD pipeline?

* Concepts:
  + Canary deployments for new API versions
  + Feature toggles
  + Schema validation in pipeline
  + Contract testing before deployment

# ✅ 23. How do you handle file upload/download in REST APIs securely and efficiently?

* Practices:
  + Use MultipartFile for uploads
  + Validate file type and size
  + Store files in S3, Blob, or database with references
  + Stream files to avoid memory overflow

# ✅ 24. How do you audit REST API activity?

* Implementation:
  + Log request/response with user, IP, timestamp
  + Store in audit tables or Elasticsearch
  + Spring AOP for audit interceptors

# ✅ 25. How do you design REST APIs for high availability and disaster recovery?

* Strategies:
  + Load-balanced services
  + Multiple instances and replicas
  + Backup plans and health checks
  + Failover via service mesh or gateway

# ✅ 26. What status codes do you commonly use in REST APIs, and why?

* Examples:
  + **200 OK:**
  + **201 Created – resource creation**
  + **203: Not Authoritative Information**
  + **204 No Content – successful without body**
  + **400 Bad Request – invalid input**
  + **401 Unauthorized:**
  + **403 Forbidden:**
  + **404 Not Found:**
  + **409 Conflict**
  + **500 Internal Server Error**
  + **501 Not Implemented.**
  + **502 Bad Gateway.**
  + **503 Service not available.**
  + **504.Gateway timeout.**
  + **509 Bandwidth Limit Exceeded.**

# ✅ 27. How do you secure REST APIs exposed over the internet?

* Key practices:
  + HTTPS (TLS)
  + API tokens, OAuth2, JWT
  + IP whitelisting
  + Rate limiting
  + Request/response encryption for sensitive data

# ✅ 28. How do you manage secrets and configurations in REST-based applications?

* Tools:
  + Spring Cloud Config
  + HashiCorp Vault
  + Kubernetes Secrets
  + Avoid hardcoded credentials

# ✅ 29. How do you implement custom exception handling in Spring Boot REST APIs?

* Use @ControllerAdvice with @ExceptionHandler
* Return standardized error response (timestamp, message, status)
* Avoid leaking internal stack traces

# ✅ 30. How do you design REST APIs for multi-tenant applications?

* Patterns:
  + Header-based tenant identification
  + URL-based (e.g., /tenantA/users)
  + Schema-based or shared DB with tenant ID
  + Isolated authorization context per tenant

# ✅ 31. How do you manage API lifecycle (from design to deprecation)?

* Tools and processes:
  + Design: OpenAPI/Swagger
  + Testing: Postman/Newman, JUnit
  + Monitoring: Prometheus + Grafana
  + Versioning & Deprecation notices
  + API Gateway documentation portal

# ✅ 32. How do you enforce consistency in API design across teams?

* Methods:
  + API design guidelines and governance
  + Shared OpenAPI templates
  + Review process via Git PRs or central design team
  + Linting tools (like speccy)

# ✅ 33. How do you implement pagination, sorting, and filtering in REST APIs?

* Best practices:
  + Pagination: page, size
  + Sorting: sort=field,asc|desc
  + Filtering: query params (status=active&role=admin)
  + Use Spring Data + Specifications / QueryDSL

# ✅ 34. How do you implement a generic error handling mechanism in REST APIs?

* Central handler using:
  + @ControllerAdvice + @ExceptionHandler
  + Standardized ErrorResponse object with:
    - timestamp, status, error, message, path

# ✅ 35. How do you support internationalization (i18n) in REST APIs?

* Concepts:
  + Accept Accept-Language header
  + Use message bundles (messages\_en.properties)
  + MessageSource bean in Spring

# ✅ 36. How do you make REST APIs cloud-native?

* Features:
  + Stateless design
  + Environment-based config via Spring Cloud Config
  + Observability (metrics, tracing, logs)
  + Health checks (/actuator/health)
  + Readiness/Liveness probes (for Kubernetes)

# ✅ 37. What is an API Gateway and what problems does it solve?

* Acts as a reverse proxy:
  + Authentication/authorization
  + Rate limiting
  + Routing
  + Response transformation
  + Metrics collection (e.g., via Spring Cloud Gateway, Kong, or AWS API Gateway)

# ✅ 38. How do you deal with eventual consistency in REST APIs?

* Strategies:
  + Event-driven updates (Kafka)
  + Use status polling or webhooks
  + Design APIs to reflect pending status (processing, accepted)

# ✅ 39. How do you optimize performance for high-throughput REST APIs?

* Techniques:
  + Connection pooling (Apache HttpClient/WebClient)
  + Async/non-blocking IO (Spring WebFlux)
  + Database indexing and query tuning
  + Response compression (gzip)
  + In-memory caching (Caffeine, Redis)

# ✅ 40. How do you write contract tests for REST APIs?

* Use tools like:
  + **Pact** for consumer-provider contract validation
  + **Spring Cloud Contract** to auto-generate tests
  + Integrate contract tests in CI pipeline

# ✅ 41. What are some challenges in exposing REST APIs to mobile clients?

* Challenges & solutions:
  + Network latency → Minimize payload
  + Offline support → Caching or local DB sync
  + Security → Use OAuth2 + token rotation
  + Versioning → Strict backward compatibility

# ✅ 42. How do you prevent over-fetching and under-fetching in REST?

* Solutions:
  + Use pagination + filtering
  + Consider GraphQL for dynamic queries if needed
  + Customize response DTOs

# ✅ 43. How do you handle long-running jobs in REST APIs?

* Techniques:
  + Asynchronous APIs (return job ID)
  + Polling endpoint /status/{jobId}
  + Use messaging queues for backend processing

# ✅ 44. How do you implement audit trails for sensitive REST operations?

* Strategy:
  + Intercept critical endpoints
  + Log user, action, resource ID, timestamp
  + Store audit logs separately or send to Elasticsearch/Kibana

# ✅ 45. How do you handle sensitive fields in the API response (e.g., passwords, tokens)?

* Approach:
  + Never expose sensitive fields (use DTOs)
  + Use JSON Views or filters to exclude fields
  + Secure logging by masking or skipping sensitive data

# ✅ Implementation of Swagger.

1**. Add Springdoc OpenAPI dependency**

**2️. Create REST endpoints with annotations like @Operation**

**3️. Run app and access /swagger-ui.html**

**4️. Use /v3/api-docs to fetch raw JSON spec**

**5️. Convert docs to YAML/HTML/PDF using Swagger Editor or plugins**

**6️. Optionally automate documentation export using Swagger2Markup**

# 1. Add Springdoc OpenAPI dependency

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

# 2. Create REST endpoints with annotations like @Operation

* @Operation(summary = "Returns a Hello message")
* @GetMapping("/hello")
* public String hello() {
* return "Hello, Swagger!";

# 3. Run App and Access /swagger-ui.html

<http://localhost:8080/swagger-ui.html>

# 4. Use /v3/api-docs to Fetch Raw JSON Spec

This URL returns the raw OpenAPI specification in JSON format,

automatically generated from your controller classes.

http://localhost:8080/v3/api-docs

# 5. Convert Docs to YAML/HTML/PDF Using Swagger Editor or Plugins

Tools you can use:

* Swagger Editor → Online tool to import JSON and export YAML or print to PDF
* Swagger2Markup + AsciiDoctor → Converts JSON to HTML/PDF during build

# Q. When you are used Set interface in your project?

* To store unique vehicle registration numbers.
* To track unique policy types.
* To collect unique claim IDs and manage unique customer IDs
* To ensure no duplicate user email or contact numbers

# Q. When you are used Set interface in your project?

* To store a list of policies for each customer
* To manage a customer’s payment history
* To display search results like matching vehicles or claims

# Q. When you are used Map interface in your project?

* To map customer ID to customer details
* → Helps quickly find customer info using a unique ID as the key.
* To link vehicle registration number with its policy
* To track claim IDs with their status (e.g., pending, approved)
* To store usernames and passwords (for login).

# Q. where you are used threads in your vehicle insurance project?

* "I used threads in my project to handle background tasks like sending notifications,
* processing claim approvals, and generating reports without blocking the main application flow."

Sending Email/SMS Notifications

→ Threads were used to send policy renewal or claim status emails in the background after a trigger (like claim update).

new Thread(() -> sendEmail(customerEmail)).start();

Processing Insurance Claims in Background

→ Claims that required validation (e.g., image/document analysis) were processed using threads to keep the UI responsive.

Generating Downloadable Reports

→ Large reports (like all claims for a month) were generated in a separate thread so users could continue using the system.

Scheduled Policy Expiry Checks

→ A thread ran daily to check for policies about to expire and triggered notifications accordingly.

Uploading Large Files (e.g., claim documents or images)

→ Upload operations were offloaded to separate threads to prevent the main app from freezing.