

REST API Interview Questions and Answers

Intern to senior level

REST API interview questions and answers with explanations and examples.

3 questions Read online: <https://www.fredevtemplates.com/interview-questions/rest-api>

1. What is the difference between REST, GraphQL and gRPC?

System Design Medium Mid-level, Senior

REST exposes resources over HTTP URLs and is simple and cache-friendly, GraphQL lets clients ask for exactly the fields they need from a single endpoint, and gRPC uses binary Protocol Buffers over HTTP/2 for fast service-to-service calls.

REST is the default for public APIs: it uses standard HTTP methods and status codes, works with browser and CDN caching and is easy to debug. Its weaknesses are over-fetching and under-fetching, which can force clients to make several round trips.

GraphQL solves that for complex front ends, such as mobile apps needing different shapes of data, but caching is harder and you must guard against expensive queries with depth and cost limits. gRPC gives strongly typed contracts, streaming and low latency, which suits internal microservices, but browsers need a proxy to call it directly.

2. What is idempotency in REST APIs and why does it matter?

System Design Medium Mid-level, Senior

An operation is idempotent if repeating it any number of times has the same effect as doing it once; it matters because network retries can otherwise create duplicate orders or payments.

In HTTP, GET, PUT and DELETE are defined as idempotent, while POST is not. Clients and gateways retry requests after timeouts, so a non-idempotent payment endpoint could charge a customer twice.

The standard fix is an idempotency key: the client sends a unique key in a header, the server stores the key with the result of the first request, and any repeat with the same key returns the stored result instead of running again. Store the key and the business change in the same transaction, and expire old keys after a sensible period.

```
POST /payments
Idempotency-Key: 7f3c1a2e-9d41-4b6e-8a53-0c1f6d2b9e11
Content-Type: application/json

{ "orderId": 1042, "amount": 4999 }
```

3. How do you secure a Node.js Express API?

Node.js Hard Mid-level, Senior

Validate all input, use parameterised queries, add security headers with helmet, rate-limit requests, restrict CORS, hash passwords, keep secrets in environment variables, and keep dependencies updated and audited.

Input: validate and sanitise every body, query and parameter with a schema library such as Zod or Joi, and never build SQL or shell commands by string concatenation, which prevents injection. Output: do not send stack traces or internal error messages to clients.

Transport and headers: use HTTPS, set helmet() for secure headers, configure CORS to allow only your own origins, and set cookies as HttpOnly, Secure and SameSite. Abuse: add rate limiting (for example express-rate-limit), limit request body size and add stricter limits on login and password reset endpoints. Operations: store secrets in environment variables or a secret manager, run npm audit, pin dependency versions with a lock file, run the process as a non-root user and log security events.

```
import helmet from "helmet";
import rateLimit from "express-rate-limit";
import cors from "cors";

app.use(helmet());
app.use(cors({ origin: ["https://www.example.com"] }));
app.use(express.json({ limit: "100kb" }));
app.use("/api/login", rateLimit({ windowMs: 15 * 60 * 1000, max: 10 }));
```

These answers are written for real interview conversations. Prefer explaining a trade-off out loud over reciting a definition. Full explanations and related questions: <https://www.freedevtemplates.com/interview-questions/rest-api>