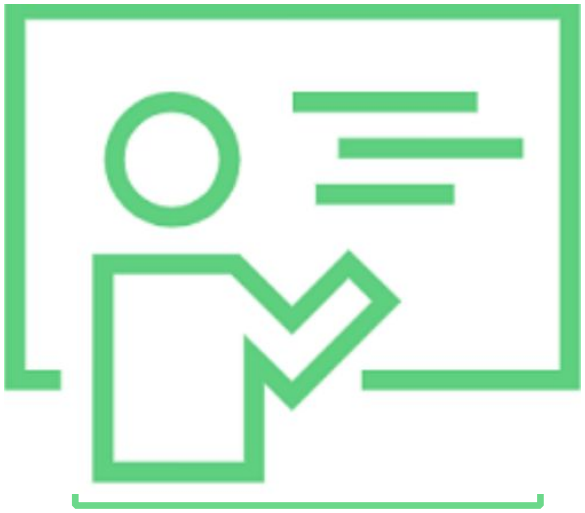


API-konsumtion och robusta klienter

Joakim Englund, Systementor AB

Repetition från dag 10

- Baslinje före förändring
- Hypotes före verktyg
- Anslutningsfel saknar HTTP-status
- HTTP 400 och 500 ligger på olika felgränser

Dagens mål

- Validera status, innehållstyp och data
- Hantera timeout med tydlig budget
- Skilja permanenta och tillfälliga fel
- Göra begränsade och säkra retries
- Dokumentera klientens felpolicy

Kontroller

Ett bra första steg vi redan tittat på är att kontrollera i flera steg.

Transport: Status → Content-Type: Parsning → Schema: Semantik

Att ett steg är okej garanterar inte att nästa är det.

Statuskod är del av kontraktet

Klass	***Betydelse***
2xx	Begäran accepterades eller lyckades
4xx	Klienten behöver ändra något
5xx	Tjänsten misslyckades

200 räcker inte

- Kontrollera `Content-Type`
- Parsa med felhantering
- Kontrollera obligatoriska fält
- Kontrollera typer och enheter
- Avvisa orimliga värden

Timeout betyder okänt utfall

Klient: timeout
Server: kan ha gjort ingenting, börjat eller redan blivit klar

Fyra budgetar

- Anslutningstid
- Tid att läsa svaret
- Antal försök
- Total tid för hela operationen

Retry är ett beslut

Felklass + operationens semantik + budget → retry eller stopp

Retry är inte standard för alla fel.

När retry normalt är fel

- 400 med samma payload
- 401 med samma token
- 403 utan ändrad behörighet
- 404 på fel API-endpoint
- Icke-idempotent POST utan dublettskydd

När retry kan vara rimligt

- Tillfälligt anslutningsfel
- 429 efter angiven väntetid
- 502, 503 eller 504
- Timeout för säker/idempotent operation

Exponentiell backoff

Första försöket → 100 ms → 200 ms → 400 ms → stopp

- Medvetet introducerat jitter sprider klienterna i tid
- Maxgräns hindrar orimlig väntan

Respektera Retry-After

HTTP/1.1 429 Too Many Requests
Retry-After: 1

Servern anger tidigaste lämpliga nytt försök antingen i sekunder eller med ett datum.

Idempotens skyddar mot dubletter

Idempotens betyder att upprepade anrop inte ändrar något. När det gäller HTTP är det bara POST och CONNECT som inte ses som idempotenta.

- GET, PUT, DELETE, osv ska inte ändra serverns tillstånd
- POST kan behöva `Idempotency-Key`

Felmeddelanden för program och människa

{"code":"invalid_value","message":"Value must be finite"}

- Stabil statuskod styr programmet
- Begriplig text hjälper människan

Logga beslut, inte hemligheter

Logga:

- `request_id`, försök och felklass
- Väntetid och slutligt beslut

Logga inte:

- Token, lösenord eller privat nyckel
- Felaktigt lösenord ska inte loggas då det kan vara väldigt nära ett korrekt lösenord

Demo: permanent fel

scenario=unauthorized → HTTP 401 → stoppa
scenario=bad-request → HTTP 400 → stoppa

Demo: tillfälligt fel

scenario=flaky → 500 → vänta → 200
scenario=rate-limit → 429 → Retry-After → 200

Demo: trasigt lyckat svar

HTTP 200 + application/json + trasig JSON

- Transporten lyckades
- Klienten måste ändå rapportera kontraktsfel

Laborationens definition av klar

- Minst sex dokumenterade scenarier
- Rätt felklass och retrybeslut
- Begränsad total budget
- Validering efter 2xx
- Token/lösenord finns inte i loggen
- Policy kopplad till eget IoT-case