Before build starts

API requirements & integration design — the analyst's craft.

Marko Hartikainen 4 August 2026

TODAY'S VILLAIN · A REAL SHAPE OF PAIN

Four hours. Six percent typos. Zero visibility.

— **Repair shops** — the workshops and dealerships reporting damage on the customer's behalf — send it by **e-mail**; someone re-types it into the core claims system.

— Median **4 hours** from report to registration · **~6%** of claims carry re-typing errors · the shops **phone in** to ask for status.

— The target: registered **within a minute**, status visible in the repair-shop portal automatically.

this case carries the whole session — every block moves it forward

FROM PART 1 · THE PROMISE WE MADE

The questions you ask in week one decide whether the integration hurts in month six.

Today is week one.

RECAP · PART 1 IN ONE SLIDE

You have a map and five questions

request / response
REST · GraphQL · gRPC · SOAP*

push
webhooks · WebSockets

messaging
queues · pub/sub · event streams

bulk
files · SFTP · EDI* · ETL / ELT

infrastructure
API gateway · iPaaS · ESB*

five routing questions: answer-to-proceed? · data or process? · volume & bursts? · other side down? · who controls both ends?

Where did your API land on the map?

- Two or three volunteers: name the API, its family, sync or async, how timely.
- Then the valuable part: **the question about it you couldn't answer.**
- Drop yours in the chat — several of them get answered live today; the rest go to the parking lot.

the question you can't answer is exactly what a requirements checklist is for

THE SERIES · YOU ARE HERE

Part 2: analyse & specify

1 · done ✓

Understand & choose

The landscape, the framework, when not to build an API.

2 · today

Analyse & specify

Discovery → requirements → design contribution → roles. Plus your first live API calls.

3 · TBA

Secure

REST security layer by layer — you spot the weaknesses.

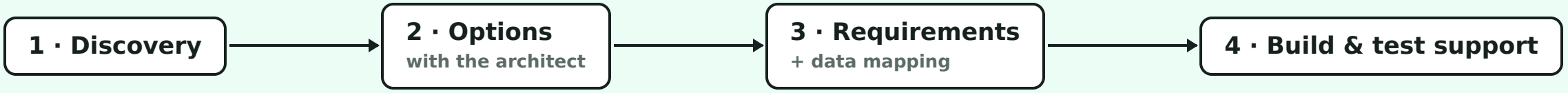
4 · TBA

Operate & work smarter

Documentation, testing, lifecycle, risks, LLMs.

plan for today: discovery → functional requirements → the contract, live → break → NFRs → diagrams & mapping → roles → gap-hunt · one 6-minute break at about :46

Four steps, and today is 1 → 3

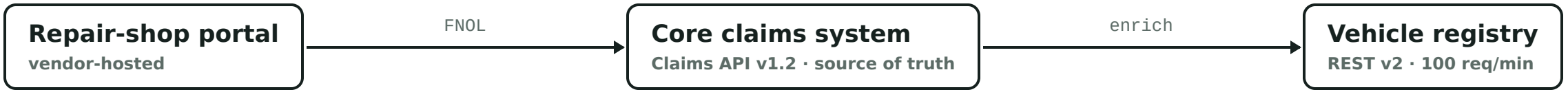


— **Discovery** — establish facts before anyone names a technology.

— **Options** — you bring the business facts, the architect brings the patterns; the choice gets *recorded with its rationale*.

— **Requirements** — the facts become testable statements: functional, non-functional, and a field-level data mapping.

The brief, and the systems behind it



+ a reinsurer receiving a nightly bordereaux file over SFTP — the bulk family never left

— The brief says: "register claims via API within a minute, show status in the portal."

— The brief does **not** say: volumes, peaks, failure behaviour, duplicate rules, who owns what.

CONCEPT · THE DISCOVERY QUESTION SET

Seven groups of questions, one page of answers

Business context

What process needs this — and what breaks if we don't build it?

Systems & ownership

Source, target, and who owns each side — technically and organisationally.

Data

Which entities and fields, source of truth, quality, personal-data classification.

Timing & volume

Freshness needed, who initiates, daily and peak volumes, growth.

Failure tolerance

What may fail silently vs must alert; is there a manual fallback?

Security & compliance

Access, legal basis for personal data, retention, audit needs.

Constraints

Partner capability, existing interfaces, deadlines, budget reality — vendor lead times bite here. Ask the API's **version & deprecation policy** now (full treatment in session 4).

The template

A discovery questionnaire — one page, seven sections, an open-questions log.

Resist solutioning

— "*REST API*" is not an answer to "*how fresh must the data be?*"

— Discovery collects **facts in business units**: minutes, counts, euros, names of owners.

— Technology re-enters at option shaping — with the architect, against the facts.

> **"We don't know yet" is a valid answer** — written in the open-questions log with an owner. An unwritten unknown returns as a month-six incident.

YOUR TURN · CHAT WATERFALL · 2 MINUTES

Your first question to this brief?

Type it in the chat — don't send. Everyone presses Enter on "3-2-1".

Onsite: shout it out when the waterfall lands.

REVEAL · WHAT DISCOVERY UNEARTHED HERE

None of this was in the brief

- ~150 reports/day sustained — but **800/day** during one storm week. Peak, not average, sizes the design.
- The reporter **waits for the claim number** → intake is synchronous. Status updates? **≤ 5 minutes is fine** → events, not polling.
- Intake **must not fail** when the vehicle registry is down.
- Duplicates are **unacceptable** — a claim registered twice is paid twice.
- The repair-shop portal's vendor supports REST and webhooks — **no message-broker client**. Constraint beats preference (question 5!).

FR anatomy: five parts, every time

- **Trigger** — what starts it (a user action, an event, a schedule).
- **Operation** — which API operation, referencing the spec by name.
- **Rule** — the validations and business rules that apply.
- **Outcome** — what happens on success, including what the user sees.
- **Unhappy path** — what happens when it fails. The part that's usually missing.

a requirement missing its unhappy path is half a requirement

SPOT THE ERROR · 60 SECONDS — WHAT'S WRONG WITH THIS FR?

requirement · draft

```
FR-X: The system prevents duplicate claims.
```

— Take a moment. Which of the five parts are present?

— Who is "the system"? What does "prevents" look like at runtime? How would a tester verify it?

The improved version

● requirement · FR-2 · duplicate protection

Trigger: a repair shop submits an FNOL from the portal
Operation: POST /claims (Claims API v1.2) with an Idempotency-Key header, one key per FNOL, reused on retry
Rule: resubmission with the same key SHALL NOT create a second claim; same vehicle+incident with a different key → 409 DUPLICATE_CLAIM
Outcome: the portal shows the returned claimId
Unhappy: on timeout the portal retries with the SAME key — no duplicate is possible by construction

THE SPINE QUESTION

Ask every arrow: "what happens when it fails?"

Every answer is a requirement. Silence here is where incidents are born.

The fallback branch is a business decision

— **FR-5:** no registry answer within **3 s** (or any error) → register *without* enrichment, flag `ENRICHMENT_PENDING`, retry up to **24 h**, then a worklist item for a human.

— Who chose 3 seconds, 24 hours, and "register anyway"? **The business did** — the analyst asked.

One operation, three views

raw · OpenAPI YAML

The source

Machine-readable contract: paths, schemas, required fields, error responses. What tools — and AI assistants — read.

rendered · Swagger UI

The browsable version

The same file, rendered: expandable operations, examples, and a **Try it out** button. Where analysts usually live.

written · wiki page

The context layer

What YAML can't say: ownership, business rules in prose, support contacts, runbook links. Your team's wiki does this job.

all three describe POST /policies of the demo API — full documentation treatment in session 4

DEMO · HANDS-ON — EVERYONE

Watch for: 200 → padlock → 401

— Open `api.cybernotes.it/mtpl/docs` — the demo API's Swagger UI.

— First call succeeds for everyone. The second one won't. Predict why before you press it.

Everything to follow along — base URL, credentials, a paste-ready cURL sheet, and the Postman/Bruno collection — is on one page: `api.cybernotes.it/mtpl/start`. Pick the client path that fits your machine.

Your first live call

- In Swagger UI: `GET /coverage-options` → **Try it out** → Execute.
- Read it like part 1 taught: verb · path · status · body.
- Postman: `GET api.cybernotes.it/mtpl/v1/coverage-options`

expected response

```
HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-Remaining: 59

{ "options": [
    { "code": "MTPL-STD",
      "name": "Mandatory motor liability",
      "term": "P1Y" } ] }
```

static fallback — if the live call misbehaves, this is what it returns

Now try to create a policy

— `POST /policies` → Try it out → Execute. Note the **padlock icon** next to it.

— Prediction was right? **401** — the passport problem. No token, no entry.

— The error body is machine-readable — remember its shape; session 4 names the standard.

expected response

```
HTTP/1.1 401 Unauthorized
Content-Type: application/problem+json

{ "type":  ".../errors/missing-token",
  "title": "Authentication required",
  "status": 401,
  "detail": "Provide a bearer token from
             POST /auth/token." }
```

static fallback · this padlock stays locked until session 3

What did you just read?

— Which **verb and path** did each call use?

— Which **status family** did each answer belong to — and whose problem is each family?

— What did the **401 body** tell you to do next?

if all three are easy, the part-1 reading skill has landed · break after this — back in 6

BREAK

6 minutes.

Back at :52 — non-functional requirements, the part where integrations actually fail.

Stretch. Water. Do not read the next slide.

NFRs: numbers, not adjectives

WEAK (AN ADJECTIVE)	STRONG (A NUMBER A TESTER CAN VERIFY)
"must be fast"	p95 ≤ 2 s at 50 req/min, including the registry lookup
"handle the load"	150/day sustained · 800/day storm peak · 100/hour target
"always up"	99.5% monthly · maintenance window Sun 02–04
"log everything"	every registration traceable by traceId + source + timestamp · kept 7 years

p95 = the time 95% of calls beat — one slow call in twenty may be slower

YOUR TURN · REWRITE DRILL · 2 MINUTES

Rewrite: "status updates must be reliable."

Make it a number with a unit and a condition — for *this* case.

Hint: reliable against what — loss? delay? disorder? Pick, then quantify.

Write down who notices

- **Alerting:** error rate > 2% over 5 minutes → claims-ops on-call · event silence > 15 minutes in business hours → same. Named conditions, named receivers.
- **Correlation:** every registration traceable end-to-end by a `traceId` — the thread support pulls when something breaks at 02:00.
- **Audit:** who registered what, from which source, when — retained 7 years. In a bank, this is a requirement, not a nice-to-have.

THE DIVISION OF LABOUR

Numbers come from the business. Feasibility comes from the architects.

Your job is to ask until the adjective becomes a number.

Context + sequence — with the else branches

- **Context diagram** — one box per system, one arrow per flow: *who talks to whom about what*. Slide 8 was one.
- **Sequence diagram** — one scenario over time, *including* what happens on timeout, rejection, retry.
- Informal is fine. Complete is mandatory — the happy path alone is a brochure, not a design.

> **If you can't draw the else branch, you've found your open question.** The diagram is a discovery tool wearing a deliverable's clothes.

Mapping finds what nobody asked

gap 1 · the enum with no home

Registry says `fuel_type: gas`. The core system has no such value. A bulk "lowercase→UPPERCASE" rule would map it *wrong, silently*. Value-by-value mapping catches it → decision needed, **owner: product**. Don't guess.

gap 2 · the field that became a requirement

Registry carries `status: stolen` — the core system has nowhere to put it. Ask "do we care?" → claims handlers *very much do* for THEFT claims → a brand-new FR routes them to manual review. **A bulk mapping would have dropped it — and the requirement with it.**

also: null ≠ absent ≠ empty string — known-empty · not-supplied · supplied-but-blank are three different facts · full method in the try-at-home exercise

You bring · they bring

YOU BRING (BUSINESS FACTS)	THE ARCHITECT BRINGS
freshness needed · volumes, peaks, growth	integration type & pattern
failure tolerance · manual fallback reality	middleware, technology, reuse
data ownership & classification	security mechanics
cost of delay — what an hour of downtime costs	effort, feasibility, platform fit

architect-side vocabulary you'll hear: **orchestration** (one coordinator drives the steps) vs **choreography** (each service reacts to events, no central conductor) — recognize the words, let them own the choice

expect 2–3 options back · the choice is recorded with its rationale — an unrecorded decision gets re-litigated

Contact · consult · inform

contact — work with directly

Business Owner · Product Owner · System Owner (**each end!**) · Data Owner · Developer/Tech Lead · Vendor contact · the counterpart's analyst

consult — at decision points

Solution / Integration / Enterprise Architect · Security Architect · IAM · DPO · Compliance/Risk · Network · API Platform Owner · QA Lead

inform — they inherit it

Operations/Support · Monitoring Owner · Release Manager — before build ends, not after go-live

the full one-pager with entry moments carries all twenty roles

The expensive late arrivals

- **The vendor's technical contact** — sandbox access, credentials, onboarding: ask in week one; these lead times routinely beat the build time.
- **The DPO** — the legal-basis question for personal data. You ask it; they answer it. Asking late can reshape the payload design.
- **The second System Owner** — every integration has *two* ends. The far end's owner learns about the project surprisingly late, surprisingly often.

the cheapest meeting is the one in week one

EXERCISE · PAIRS · ~7 MINUTES HUNTING

Gap-hunt: a draft spec landed on your desk

— A vendor sends the **v0 draft** of the policy API you're about to integrate with. Something's off. Several things, actually.

— Pairs — mix business + system analysts, room + remote. Hunt with today's checklist: discovery groups, FR anatomy, NFR numbers.

— Score: **one point per gap** phrased as a *question to the API owner*. Complaints score zero.

`spec: api.cybernotes.it/mtpl/v0/openapi.yaml · key excerpt on the next slide · six gaps are planted — find five and you're dangerous`

mtpl-openapi-v0.yaml · excerpt 1

```
paths:
  /coverage-options:  get → 200
  /auth/token:        post → 200
  /policies:          post → 201 · get → 200
  /policies/{id}:     get → 200, 404 · put → 200
  /policies/{id}/cancel: post → 200

components: (no securitySchemes)
```

excerpt 2 · the request schema

```
PolicyRequest:
  required: [vehicleId, holderName]
  properties:
    vehicleId:
      type: string
      example: "WVWZZZ1JZXW000001"
    holderName:
      type: string

POST /policies example body:
  vehicleId: "ABC-123"   # ...wait.
```

hunt with the checklist: capability? data clarity? errors? security? volumes? — one question per gap

REVEAL · THE SIX PLANTED GAPS

#	GAP	THE QUESTION IT BECOMES
1	no auth documented (yet a token endpoint exists)	how do callers authenticate — and which operations need what?
2	error responses missing almost everywhere	what does a rejection look like — codes, format?
3	vehicleId: reg number and VIN mixed in examples	which identifier, which format, validated how?
4	one-active-policy rule invisible	what happens on the second sign-up — 409? merge? silence?
5	no pagination on GET /policies	what happens at 60+ policies — and what's the page limit?
6	no rate-limit information	what are the limits, and what do we do on 429?

the live v1 in Swagger UI documents all six — that's the before/after of today's whole craft

Today moves to your checklist — the 401 waits for part 3

right now

Your checklist grows

Discovery groups, FR anatomy, NFR numbers, mapping method, SA handover, the roles map — plus your top-3 gaps from the hunt.

link → on the training page

try at home · pick one

Aim it at your API

1 · Run the discovery questionnaire + checklist against your homework API — especially the question you couldn't answer. 2 · Repeat today's two calls yourself — everything you need is the one-page **api.cybernotes.it/mtpl/start** (Postman collection linked there). 3 · Full mapping exercise with the sample specs.

for part 3

The padlock

We left `POST /policies` answering **401**. Part 3 opens it: tokens, scopes, and every control between a caller and a bank's data.

What a complete pack contains

1	Overview & context diagram — who talks to whom, one page
2	Parties & environments — owners both ends, sandbox/prod, test-data rules
3	Functional requirements — trigger · operation · rule · outcome · unhappy path
4	Data mapping — target-first field table, enum maps, gap log with owners
5	Non-functional requirements — numbers with units and conditions
6	Error handling & reconciliation — else branches, retries, source of truth
7	Security & data protection — access, classification, legal basis (part 3 deepens)
8	Acceptance criteria & test scenarios — the four families (part 4 deepens)
9	Open items — every unknown, an owner, a date

sections 7 and 8 grow in parts 3 and 4

APPENDIX · JARGON DECODER — PART 2 TERMS

FNOL	first notice of loss — the first report of a claim (here: filed by repair shops via the portal)
discovery	establishing facts (process, data, volumes, tolerances) before design
FR · NFR	functional / non-functional requirement — what it does / how well it must do it
idempotency key	one key per business action, reused on retry — makes duplicates impossible
p95	the value 95% of calls beat — honest "fast", unlike an average
traceId	correlation identifier carried through every system a request touches
enum	a closed list of allowed values — map them value-by-value, never in bulk
gap log	the mapping's list of unmappables — each entry owned and dated
system of record	the one authoritative source of a data item; everyone else holds copies
source of truth	same idea as system of record, said about a whole domain
sandbox	a safe non-production copy of an API to test against
SA	Solution Architect — patterns and feasibility; your option-shaping partner
DPO	Data Protection Officer — answers the legal-basis question you must ask
IAM	Identity & Access Management — the team behind accounts, scopes, machine identities
bordereaux	the periodic claims/premium report file an insurer sends its reinsurer
Swagger UI	the browsable, clickable rendering of an OpenAPI contract
problem+json	the standard machine-readable error format you met in the 401 — part 4 names it properly
open-questions log	every unknown written down with an owner and a date — discovery's safety net

not presented live · part 1's decoder covers the landscape terms · part 3 adds the security vocabulary