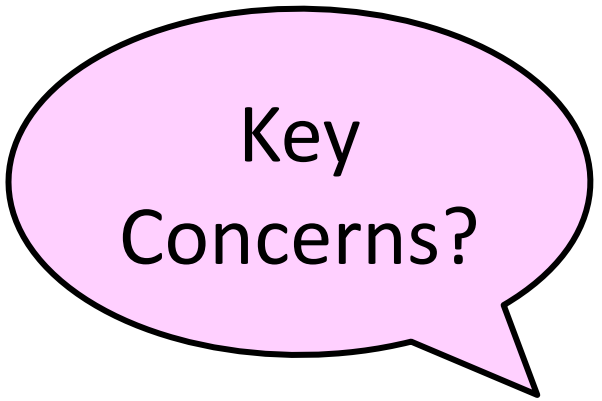
TOP REST API DESIGN PITFALLS

BY VICTOR RENTEA

Victor Rentea

- **18y of Coding**, Java Champion
- **Domains**: 🏦 🛒 📡 🚂 📣 🎰 🏥 🏛 ...
- **10y of Training** at **150** companies
- **100+ Conference Talks** on YouTube
- **Live Webinars for my Community**
- **Life** += 👰 + 👧 + 👨‍🦲 + 

Exposing an API



Concerns ~ REST APIs

- Auth(entication|orization)
- Privacy: GDPR/HIPAA
- Naming on URL (plural)
- Backwards compatibility, or Versioning😫
- Negotiation with the clients: upfront and on the way
- Be able to track your clients: who is using what (traceID, api-keys)
- Documentation
- Error handling
- Sub-resources? customer/{id}/address
- Encapsulation: how much of the internal state to expose? more: they can do more stuff; less: i can change internal stuff freely
- Protections: rate limiting, bulkeads, DoS
- Idempotency (can my client retry)
- Large payloads? eg: returning 1GB of data in json Content-Type: application/nljson
- PUT vs PATCH vs PUT /customer/{id}/address
- Should this metadata (eg: tenantId, userId): should it be header or payload?
  * X-Tracing-Id: 123 < standard. ------ Nov 2024

REST API

Co-author of **TCP/IP**

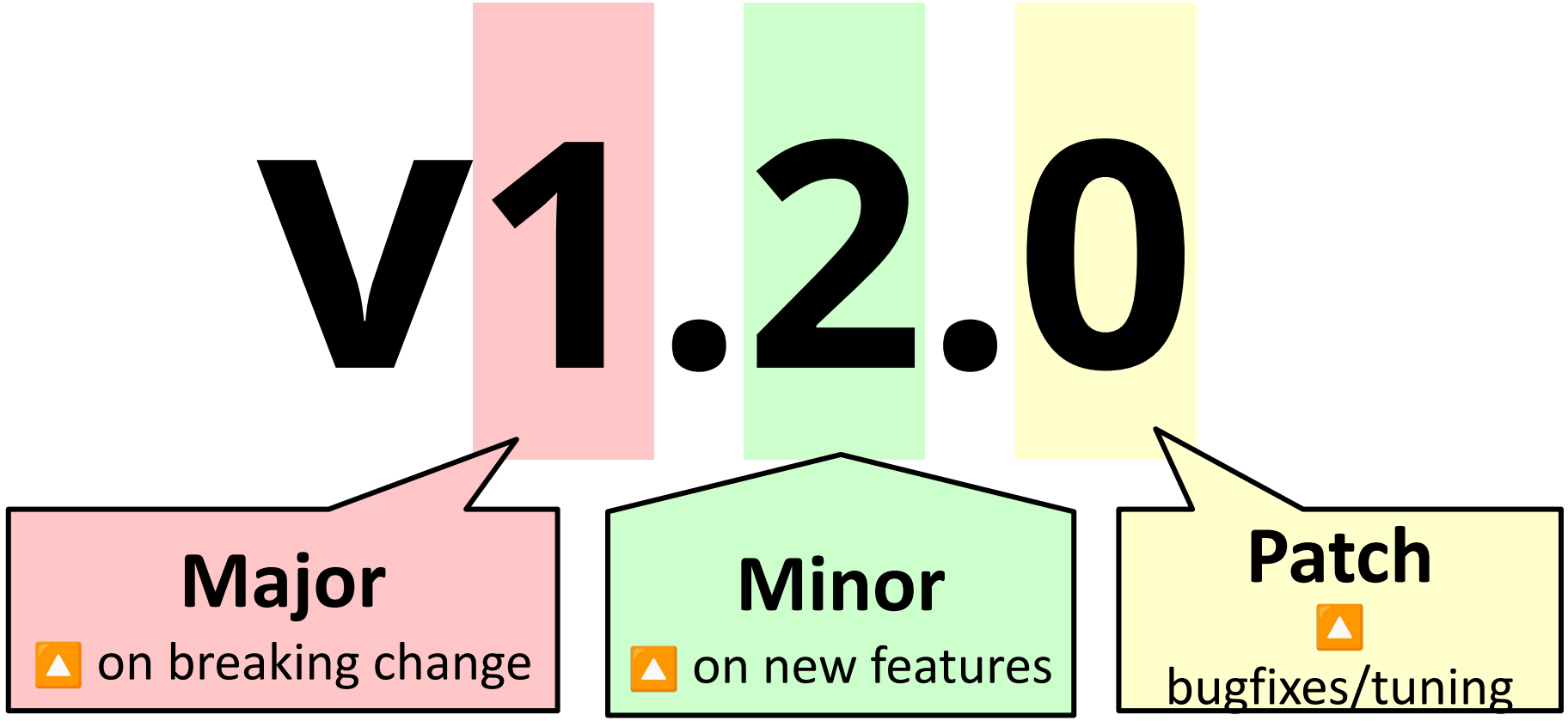
Postel's Law

Be conservative in what you do,

but liberal in what you accept from others

Semantic Versioning

SemVer.org



SemVer Exercise

Major
🔼 on breaking change

Minor
🔼 on new features

Patch
🔼
bugfixes/tuning

If today you use v1.2.0, can you switch to with no compilation errors:

a) 1.2.1 ⬆️ ✅ Yes. Probably... 🤞🍀

b) 1.3.0 ⬆️ ✅ Yes... 🤞🍀

c) 1.1.0 ⬇️ 🚫 No... if you used a 1.2 feat

d) 2.0.0 ⬆️ 🚫 No

- Semantics
- Performance

Temporary 🙏 downgrade for bugs or vulnerabilities?

Backwards Compatibility

How to Break It?

Cause a Breaking Change!

that would force clients to update

POST /users/ {email} a request

```
PUT /user/{id}/prefs

{fullName:{
   "firstName": "John",
   "lastName": "DOE",
   "emailAddress""a/com",
   "phone": ["+407129"],
   "currency":  "EUR"
}
```

*required

+regex/min len

array

+"address" *required

+"middleName"

❤️optional

One of ~~RUB~~|EUR|GBP

no longer supported

(intentional?)

Change Content-Type:
-application/xml
-application/json; charset=utf-32

a response

```
{
"date": "04/01/2023",
 age": "37"
"country": "ROU"
}+"ageStr":
```

"2023-03-01" =ISO

number

One of ~~ROU~~|ESP |NOR

❤️

can be returned

Legend

a breaking change

backwards-compatible change

🤔 Challenge enums in DTOs: eg isn't "Spain" enough?

Be Backwards-Compatible!

- Add **more optional input** fields ⭐
- Add **more output** fields ⭐

  +phone: "+40720..", (A)

- +phoneCountryCode: "+40", (B)

  +phoneLocalNumber: "720..", (B)

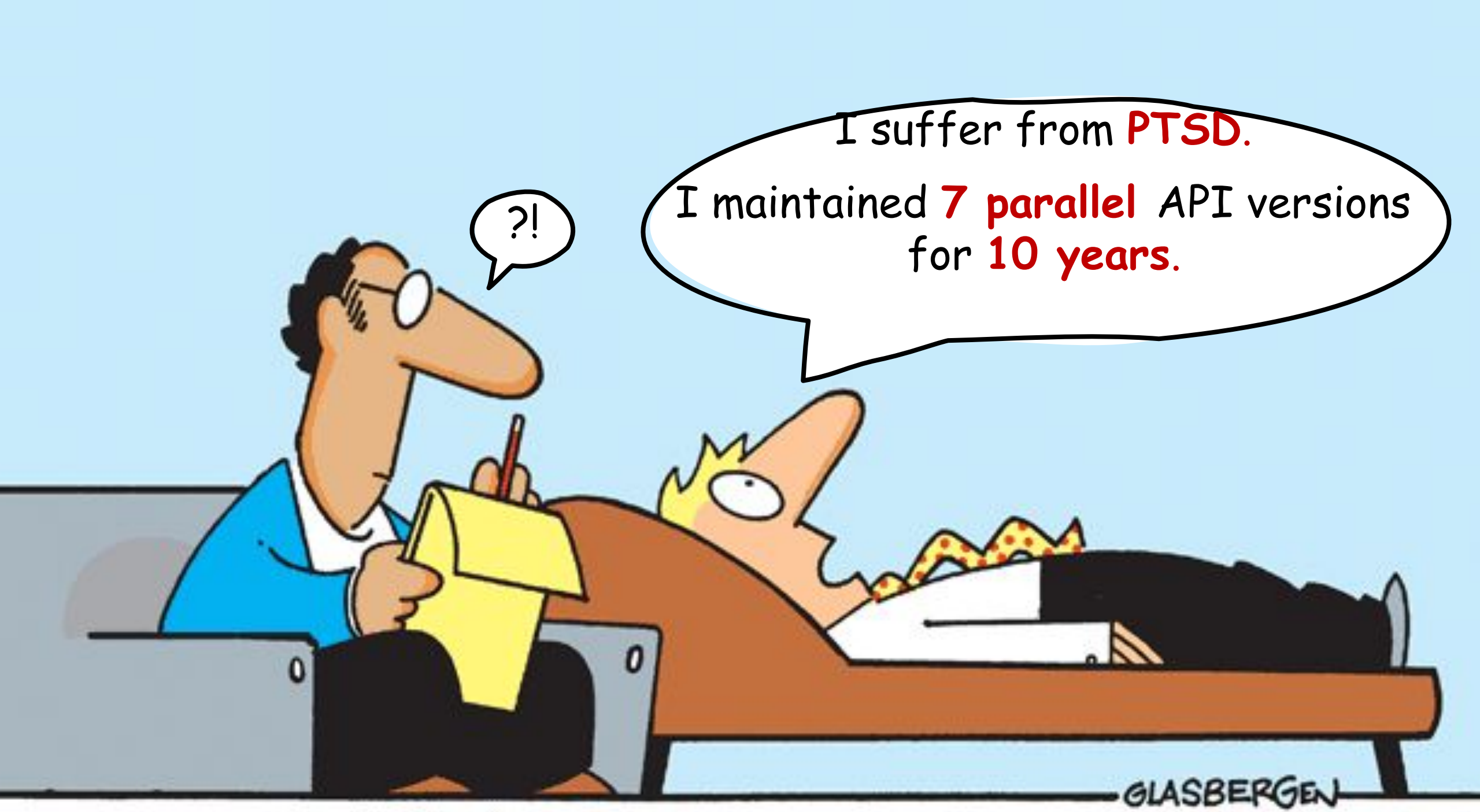
  Add **alternative representation** (B) in **response**

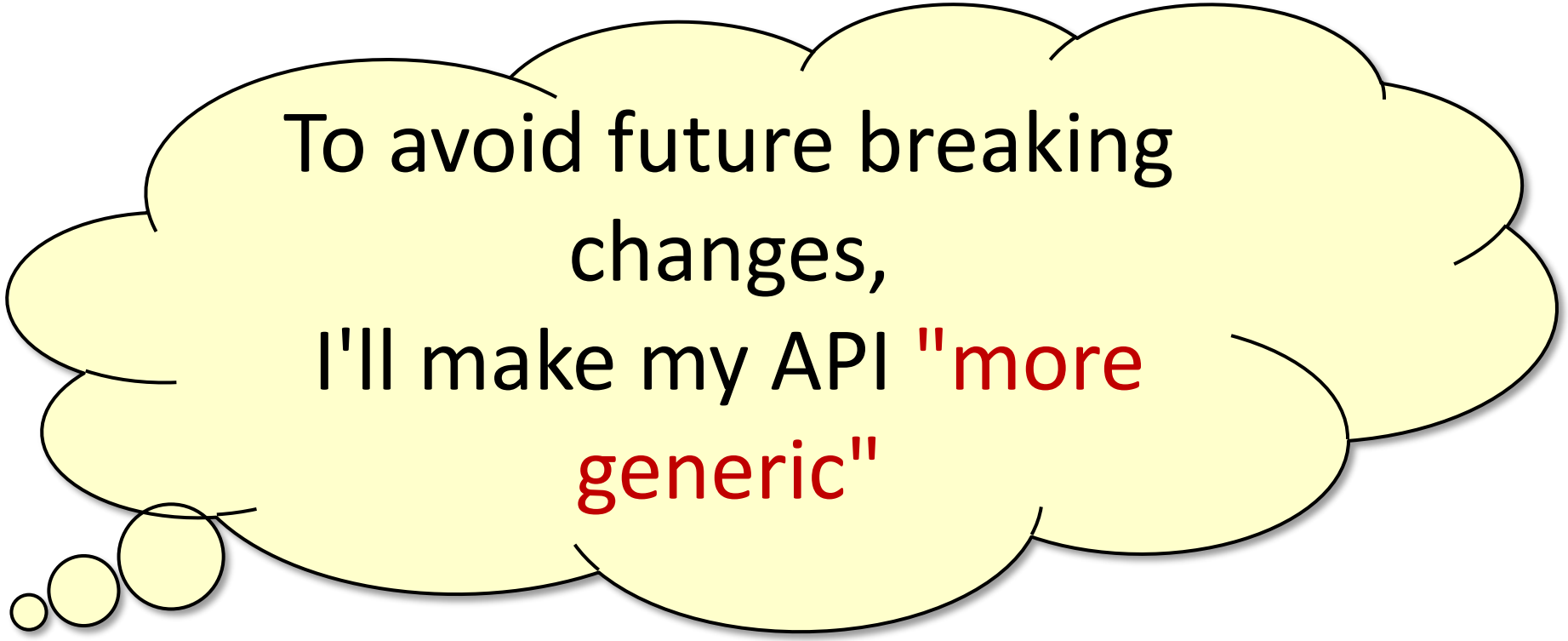
- ... or in **request – Wait!** What if a request brings both (A)+(B) ?!😱

- = **API Semantic Debt** accumulates □ clean-up in v2

v2 is Here! What next?

- Code is copy-pasted, branched❌ or riddled with if (version==2)
- **Identify v1 Clients** via request headers
  - Authorization {JWT:sub}, X-Api-Key, X-Client-Id, Service Registry, TraceID💖
- **Convince v1 Clients to Upgrade**
  - 📞, 🙏, 🤬, **No New Features**, Rate Limit↘️, Fees↗️; force mobile app update😫
  - Provide: Migration Guides, OpenRewrite recipes,... Can I code with you?🥲
  - **Don't**: ~~if (random) throw "Upgrade!"~~ or ~~sleep(1s)~~ or ~~claim 'V1 is vulnerable'!~~
- **Ideally: support max 2 parallel versions for a limited time** 🤞


I suffer from PTSD.
I maintained 7 parallel API versions for 10 years.
?!
GLASBERGEN


To avoid future breaking changes,
I'll make my API "more generic"

Future-Proof API - Looses Clarity 🤯

```
{
 emails: ["a@b.com"],

 phones: [{phone: "ABC", type:"work"}],

 metadata: [    aka 'extensions'
  {name:"age", value: 12},
  {name:"address", value:{street:..., }},
  +50 more keys added over years😵‍💫
 ] java: Map<String,?>
}
```

Perhaps **tomorrow** we start supporting more...
(but today you require and return size=1)

more *types* **tomorrow**?

more keys **tomorrow**?

To avoid future breaking changes,
I'll make my API "more generic"

👆 *these keys* are not visible in your OpenAPI = cheating on schema

Versioning Strategies

- **New microservice**: order-service-**v2** + separate (Git + CI ± **DB**)
  - Better: "corporate-order-service"
  - **Design for deletion** – easy to remove v1 later (soon! 🤞)
- **At URL root** ⭐: order-service.intra/**v2**/orders/{id}
  - "/**v1**/" is a middle finger 🖕 to your API clients, REST author Roy Fielding
- **Per-endpoint**: /orders/**v2**/{id}
  - eg in an Api Gateway exposing multiple services
- **Content-Type + Accept**: application/json+**v3**

Contract Tests

Auto-detect contract mismatch

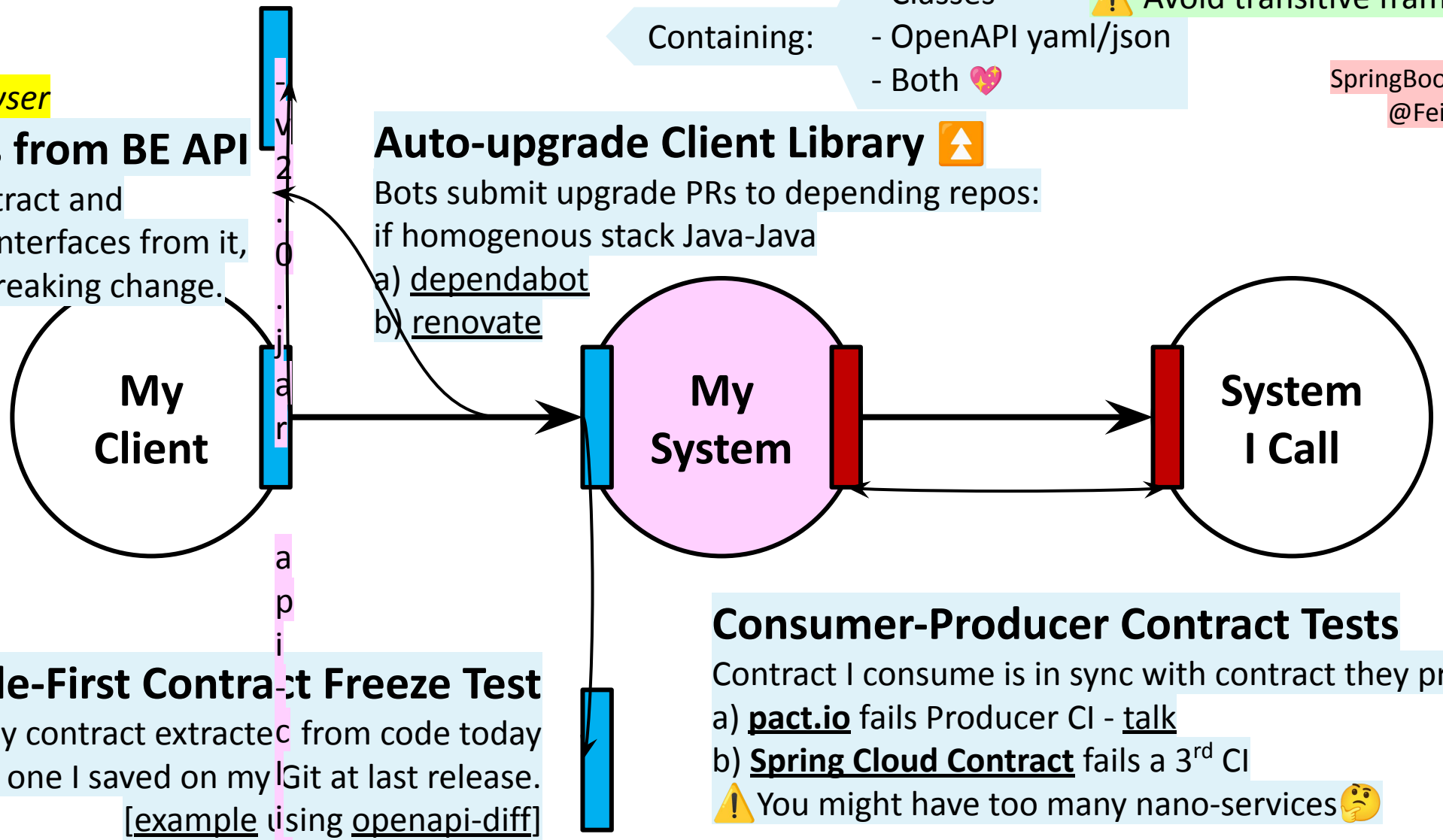
Containing:
- Classes
- OpenAPI yaml/json
- Both 💖

⚠️ Avoid transitive framework deps:
SpringBoot v2->v3
@FeignClient
guava

#musthave if client=Browser

Generate FE Types from BE API

FE build pulls the BE contract and
(re)generate TypeScript interfaces from it,
FE build fails on BE API breaking change.



Auto-upgrade Client Library ⏫

Bots submit upgrade PRs to depending repos:
if homogenous stack Java-Java
a) dependabot
b) renovate

Code-First Contract Freeze Test

My contract extracted from code today
matches the one I saved on my Git at last release.
[example using openapi-diff]

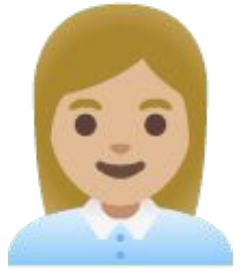
Consumer-Producer Contract Tests

Contract I consume is in sync with contract they provide.
a) **pact.io** fails Producer CI - talk
b) **Spring Cloud Contract** fails a 3rd CI
⚠️You might have too many nano-services🤔

Contract Tests

- Prevent **accidental changes in the contract I expose**
  - Compare contract extracted from code vs one saved yesterday on Git
  - Example Test using openapi-diff
- Detect a **mismatch between consumed⇔provided contract**
  - pact.io = Consumer-Driven Contract => fails the producer CI
  - Spring Cloud Contract => fails a separate CI
  - Also consider HTTP calls on CI hitting *their* staging (though flaky)
- Auto-upgrade to **newer versions of API client libraries**
  - client-service provider-api-v3.2.jar provider-service
  - renovate/dependabot submits PR if compilation succeeds upgrade to v3.3🔼

API Design



Extract contract
eg from /v3/api-docs

Generate classes at build
eg via Maven Plugin ...

Code-first

Write (Java) Dto classes

> Generate FE Dtos from BE Contract
too keep in sync

> Editable Dtos 💖
Can contain mapping/validation code

> No code generation required ⏩

Contract-first

Write (YAML) Swagger

> Mandated by protocol:
protobuf, Avro, GraphQL

> If created with business clients / py
Focus remains on contract, not implementation

> External contract to Customers (WWW)
+ documentation + examples + error codes..

⭐ Whatever strategy, develop logic outside-in from contract inwards

Performance

Get One of Many

What can go wrong?

You have many records in your DB, exposed via:

`GET /products/{id}`

🤔 Wrong Boundaries?

Clients could **network-call-in-a-loop** = **performance massacre** ☠️:

```
for (id in listOfIds) {.. yourApi.getById([id]) .. }
```

Expose a Batch Query 😈 Sequential IDs? 👉 UUIDs cannot be scanned, but

- `GET /products?id=1,2,4,7,8,..10K` ↩️ URL can get truncated at 2000 characters ⚠️
- **POST** `/products/get-many + [1,2,...100K]` keep payload size decent ⚠️ ≤1000?
- `GET /products + [1,2,4..]` GET got a body in 2021, but proxies might still drop
- 😉 Rate-limit client calls : ~~429 To~~ ⚠️ Many Requests + 'Please use the batch API'
- On error❌, which item was not found?

KISS: implement a batch API when prod metrics/traces show it's needed.

That was a **Batch Query**. But how about a...

Batch Command

POST /products/many
[{item#1}, ..., {item#10}, ...]

❌ Error Handling:
- Which payload was wrong? eg #10? Why?
- Did the others get committed? eg #1--#9?
- On error, should I stop or skip?

⏱️ Timeout: How long would this call take?

Performance & Decoupling

👉 Hand-made

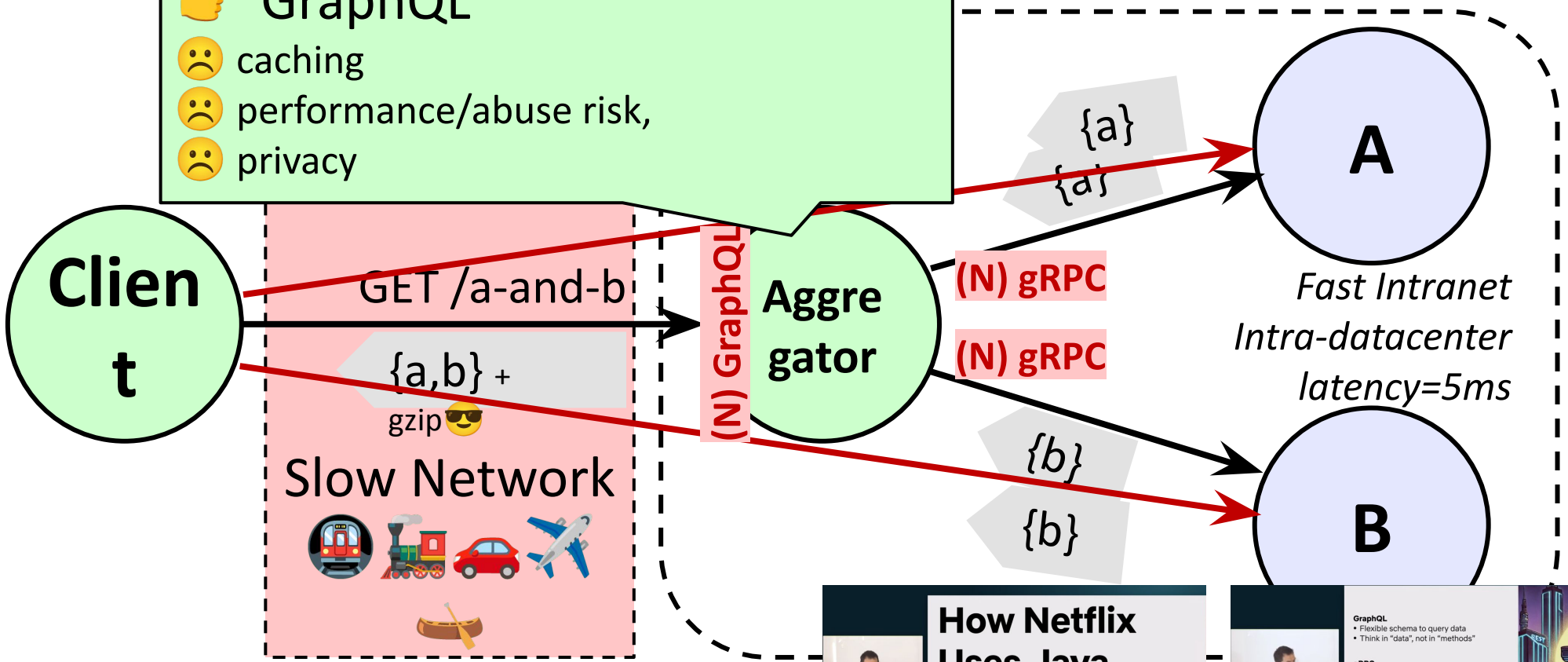
☹️ owner=FE

👉 GraphQL

☹️ caching

☹️ performance/abuse risk,

☹️ privacy



Reminder: you are NOT Netflix

Apr 25, 2025 @ JavaOne - 165k views

GraphQL ?

FE wants BE to expose GraphQL to reduce its friction with BE
=> Go FullStack! Have that FE change the BE endpoint

Various teams are interested to **READ** different sub-sets of attributes of a large structure

Canonical use-case: Facility CMS
- A building has **hundreds** of attributes: floors, address, MWh, people, costCenter, ...
- 20 client applications call that API with different needs

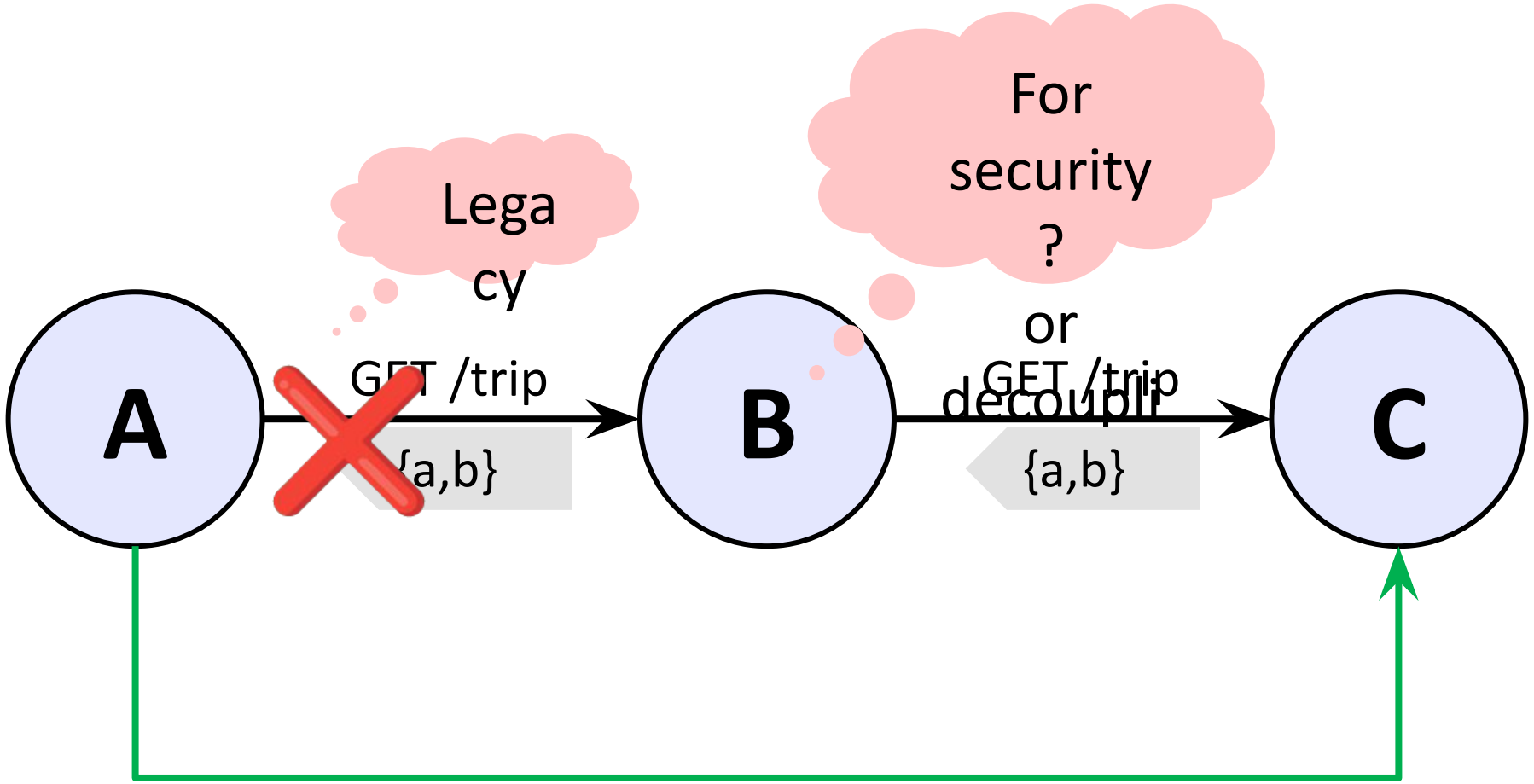
+ less versioning problems

+ send less data to clients (less traffic, eg client=mobile app)

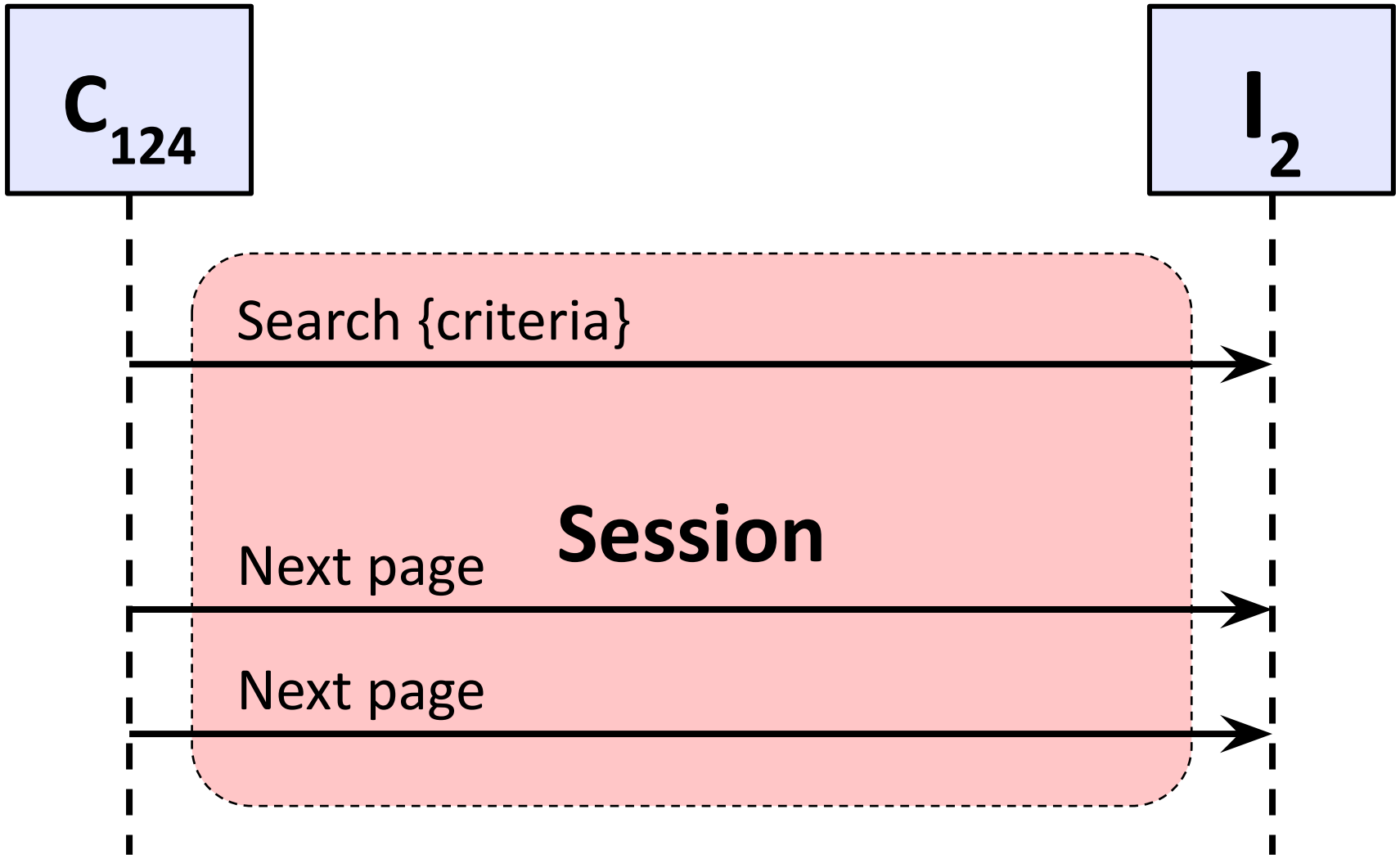
stage1: we were calling full REST APIs udner the hood

stage2: attribute resolvers that dynamically fetch stuff THE CLIENT requested.(+SELECT, +GET). = more efficient on server-side

Request Proxying



Stateful Endpoints



API design that can Hurt Performance

Network-call-in-a-loop

Huge Payloads

Proxy by accident

Stateful Endpoints

ORM Lazy Loading (goof🤤) 👉

(new features)

Performance

Complexity

Scalability

Don't Expose Internal Model in your API

```
@GetMapping // my REST API
public Customer findById(id)
```

CustomerDto

Opinions?

```
// Core Domain Model ❤️
public class Customer {

  ...
  String name;

  @JsonIgnore
  String phone;

  @JsonFormat(..)
  LocalDate birthDate;

  @OneToMany
  List<Project> projects;
}
```

❄️ **Freezes Your Domain Model** as your clients grow coupled to it

🔐 **Security Risk** to expose sensitive data

🗑️ **Pollutes Domain** with presentation

⏱️ **Performance Goof** if ORM lazy-loading

stable clear documented

Separate **Contract** (API DTO)

from **Implementation** (Domain Model)

Evolving to Simplify Logic

Data Transfer Objects □Mapping□ Domain Model

= the price to pay for decoupling the two models

Automatic Mappers

```
Dto t
   var dto = new Dto();
   dto.setName(entity.getName());
   dto.setEmail(entity.getEmail());
   dto.setPhone(entity.getPhone());
   ..... <20 more>
   return dto;
}
```

Boilerplate Code

```
var dto = Dto.builder()
  .name(entity.getName())
  .email(entity.getEmail())
  ..
.build(); // immutable DTO
```

```
val dto = Dto(//named params
  name = entity.name,
  email = entity.email,
  ...
);
```

Automatic Mappers

<< My API Model >>

```
class CustomerDto {
  firstName:string
  lastName:string
  ...
}
```

<< Domain Model >>

```
class Customer {
  firstName:string
  lastName:string
  ...
}
```

mapping happens automatically if field names match

Libraries:

- MapStruct 👑
- ModelMapper
- Dozer
- orika
- AutoMapper

Best: it generates source code:
1) can see/debug it
2) fastest (JIT-compiled)
3) simple to give up on it:
move generated -> src/main

The Pitfall of Automatic Mappers

Frozen for Clients

<< My API Model >>

```
class CustomerDto {
  FirstName:string
  LastName:string
  ...
}
```

<< Domain Model >>

```
class Customer {
  firstName:string
  lastName:string
  fullName:FullName
  ...
}
```

I'll group them in a new Value Object

Ever evolving

- **Build/Tests should fail** 🤞
- **Complex Mapping Customization** 😵‍💫

NEAH...
I'll do it next sprint

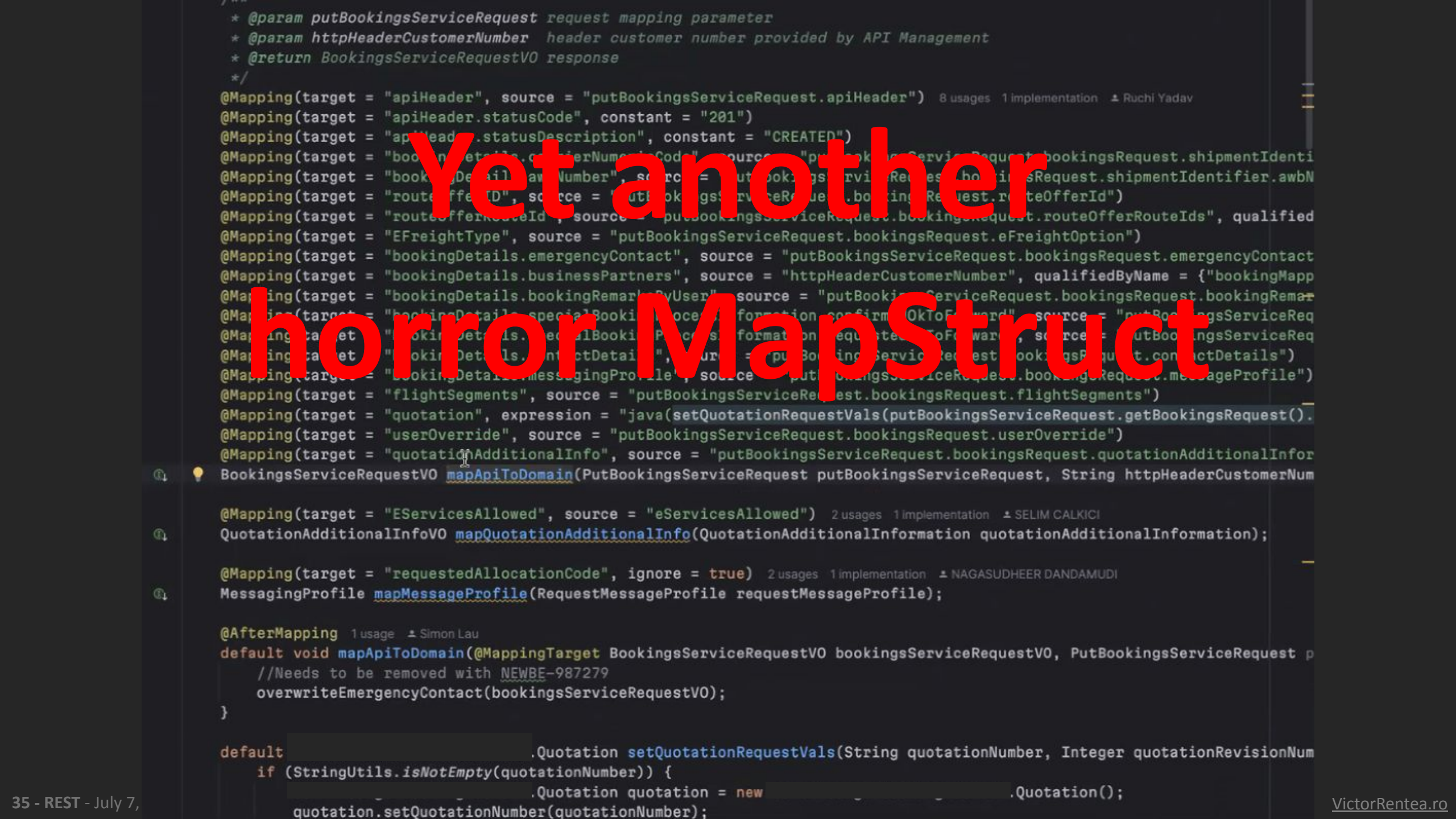
- **Tempting to keep the models in sync -** Why then DTOs?!

The Pitfall of Auto-Mappers:

Tempting to keep the API Model in sync with Domain Model.

When Auto-magic Mapping grows complex

=> Switch to **manual mapping on THAT flow**




```
 * @param putBookingsServiceRequest request mapping parameter
 * @param httpHeaderCustomerNumber  header customer number provided by API Management
 * @return BookingsServiceRequestVO response
 */
@Mapping(target = "apiHeader", source = "putBookingsServiceRequest.apiHeader")  8 usages  1 implementation  Ruchi Yadav
@Mapping(target = "apiHeader.statusCode", constant = "201")
@Mapping(target = "apiHead .statusDescription", constant = "CREATED")
@Mapping(target = "boo n etails. ierNumericCod ", source = "p ookingsServi Request.bookingsRequest.shipmentIdenti
@Mapping(target = "book Detail awNumber", sc c = ut okings rvi Re es bo i sRequest.shipmentIdentifier.awbN
@Mapping(target = "route ffe D", so ce = utBookingsS viceRe ue .bo ing Re uest.r teOfferId")
@Mapping(target = "route fferRouteId", source = "putBookingsServiceRequest.bookingsRequest.routeOfferRouteIds", qualified
@Mapping(target = "EFreightType", source = "putBookingsServiceRequest.bookingsRequest.eFreightOption")
@Mapping(target = "bookingDetails.emergencyContact", source = "putBookingsServiceRequest.bookingsRequest.emergencyContact
@Mapping(target = "bookingDetails.businessPartners", source = "httpHeaderCustomerNumber", qualifiedByName = {"bookingMapp
@Mapping(target = "bookingDetails.bookingRemarkByUser", source = "putBooki ServiceRequest.bookingsRequest.bookingRema
@Mapping(target = "bookingDetails.specialBooki roce formation confirm OkToF ward", source = "putBo ingsServiceReq
@Mapping(target = "b okin Det ils. ecialBooki P c si formation equ ste oF ar , s rce = utBoo ingsServiceReq
@Mapping(target = "b okin Det ils. ntactDetai ", sour = "putBo ingServi Re est ooki sR u t.con actDetails")
@Mapping(target = "BookingDetails.messagingProfile", source = "put ingsServiceRequest.bookingsRequest.messageProfile")
@Mapping(target = "flightSegments", source = "putBookingsServiceRequest.bookingsRequest.flightSegments")
@Mapping(target = "quotation", expression = "java(setQuotationRequestVals(putBookingsServiceRequest.getBookingsRequest().
@Mapping(target = "userOverride", source = "putBookingsServiceRequest.bookingsRequest.userOverride")
@Mapping(target = "quotationAdditionalInfo", source = "putBookingsServiceRequest.bookingsRequest.quotationAdditionalInfor
BookingsServiceRequestVO mapApiToDomain(PutBookingsServiceRequest putBookingsServiceRequest, String httpHeaderCustomerNum

@Mapping(target = "EServicesAllowed", source = "eServicesAllowed")  2 usages  1 implementation  SELIM CALKICI
QuotationAdditionalInfoVO mapQuotationAdditionalInfo(QuotationAdditionalInformation quotationAdditionalInformation);

@Mapping(target = "requestedAllocationCode", ignore = true)  2 usages  1 implementation  NAGASUDHEER DANDAMUDI
MessagingProfile mapMessageProfile(RequestMessageProfile requestMessageProfile);

@AfterMapping  1 usage  Simon Lau
default void mapApiToDomain(@MappingTarget BookingsServiceRequestVO bookingsServiceRequestVO, PutBookingsServiceRequest p
    //Needs to be removed with NEWBE-987279
    overwriteEmergencyContact(bookingsServiceRequestVO);
}

default                                    .Quotation setQuotationRequestVals(String quotationNumber, Integer quotationRevisionNum
    if (StringUtils.isNotEmpty(quotationNumber)) {
                                    .Quotation quotation = new                                .Quotation();
        quotation.setQuotationNumber(quotationNumber);
```

```java
@Builder
public class BookingsServiceRequestVO {

    /**
     * API Header in the response
     */
    private ApiHeaderVO apiHeader;
    /**
     * Return Code in the response
     */
    private ReturnCode returnCode;
    /**
     * The efreight type.
     */
    private String eFreightType;
    /**
     * The route offer id.
     */
    private String routeOfferID;
    /**
     * The routeOfferRouteId
     */
    private String routeOfferRouteId;
    /**
     * The service unit.
     */
    private ServiceUnitVO serviceUnit;
    /**
     * The quotation
     */
    private Quotation quotation;
    /**
     * This describes the booking details
     */
    private Booking bookingDetails;
    /**
     * This property describe what is the endpoint's name for this
     * request object
     */
    private Endpoint apiOperation;
    /**
     * header channel user
     */
    private String headerChannelUser;
```

Abusing the same DTO for GET + POST/PUT

```
@GetMapping("{id}")
InventoryItemDto get(id) {
```

```
@PostMapping
void create(InventoryItemDto) {
```

InventoryItemDto

```
{
 "id": null,   always null in create flow
 "name": "Chair",
 "supplierName": null,
 "supplierId": 78,
 "description": "Soft friend",
 "stock": 10,
 "status": null,
 "deactivationReason": null,
 "creationDate": null,
 "createdBy": null
}
```

Opinions?

The Contract becomes:

– misleading for **clients** 🥴

"Why should I provide the id?"

– confusing to **implement**

"Why is that field always null?"

– **couples** endpoints

get changes ==> create changes

Dedicated Request/Response Structures

*= **CQRS** at the API Level*

```
@GetMapping("{id}")
GetItemResponse get(id) {
```

```
@PostMapping
void create(CreateItemRequest){
```

GetItemResponse

```
{
 "id": 13,
 "name": "Chair",
 "supplierName": "ACME",
 "supplierId": 78,
 "description": "Soft friend",
 "stock": 10,
 "status": "ACTIVE",
 "deactivationReason": null,
 "creationDate": "2021-10-01",
 "createdBy": "Wonder Woman"
}
```

CreateItemRequest

```
{
 "name": "Chair",
 "supplierId": 78,
 "description": "Soft friend",
 "stock": 10
}
```

+validations
@NotNull..

in a 'dto' package

A shared 'dto' package could encourage reusing DTOs between endpoints = BAD PRACTICE

Keep request/response objects next to Use-Cases and separated from each other = VSA

CQRS

Command/**Query** Responsibility

Update Data

Read Data

regation

not ~~Separation~~

Command/Query Responsibility Segregation

Most people perceive software systems as *stores of records:*
that they **C**reate, **R**ead, **U**pdate, **D**elete and Search (CRUDS)

As a system grows **complex**:

READ aggregates (SUM..) or enriches the data: JOIN, API calls..

🏆 **low latency & high availability**

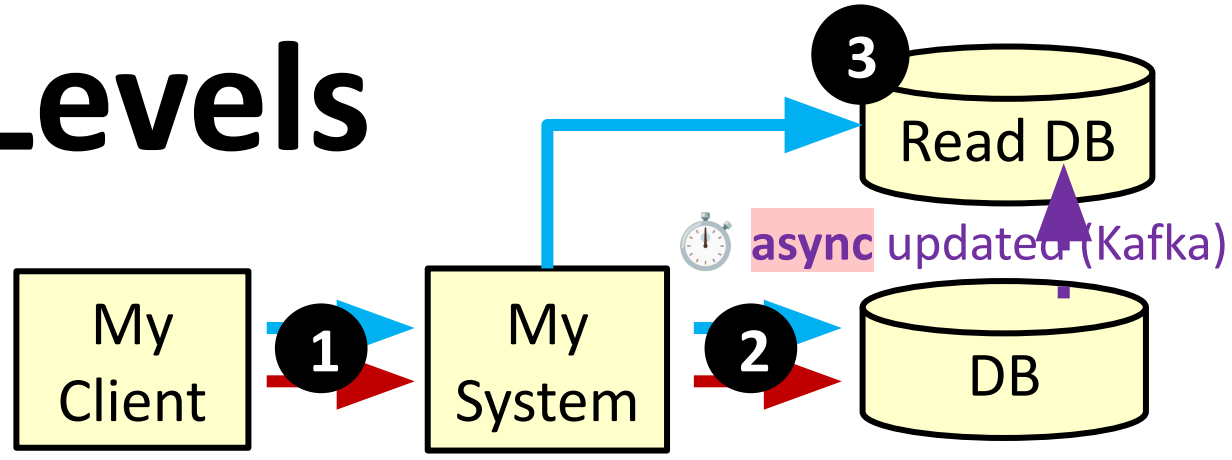
WRITE stores additional metadata: createdBy=, ...

🏆 **preserve data consistency**

CQRS = use separate WRITE / READ data models

https://www.eventstore.com/blog/transcript-of-greg-youngs-talk-at-code-on-the-beach-2014-cqrs-and-event-sourcing

CQRS Levels



1. CQRS at API level to Clarify Contract

GetItemResponse -- query

CreateItemRequest -- command

2. CQRS at SQL Interaction to Optimize Read, especially when using an ORM

Search: SELECT ~~u~~ new dto.SearchResult(u.id, u.name,..*<few>*) FROM User u ...

Updates use valid Domain Model Entities: repo.save(user) 🔼 **Strong Consistency**

---

3. Async CQRS (popularized by Greg Young)

Update a Write Storage: SQL, Event Store...

Query a Read Storage, async updated 😱🤯

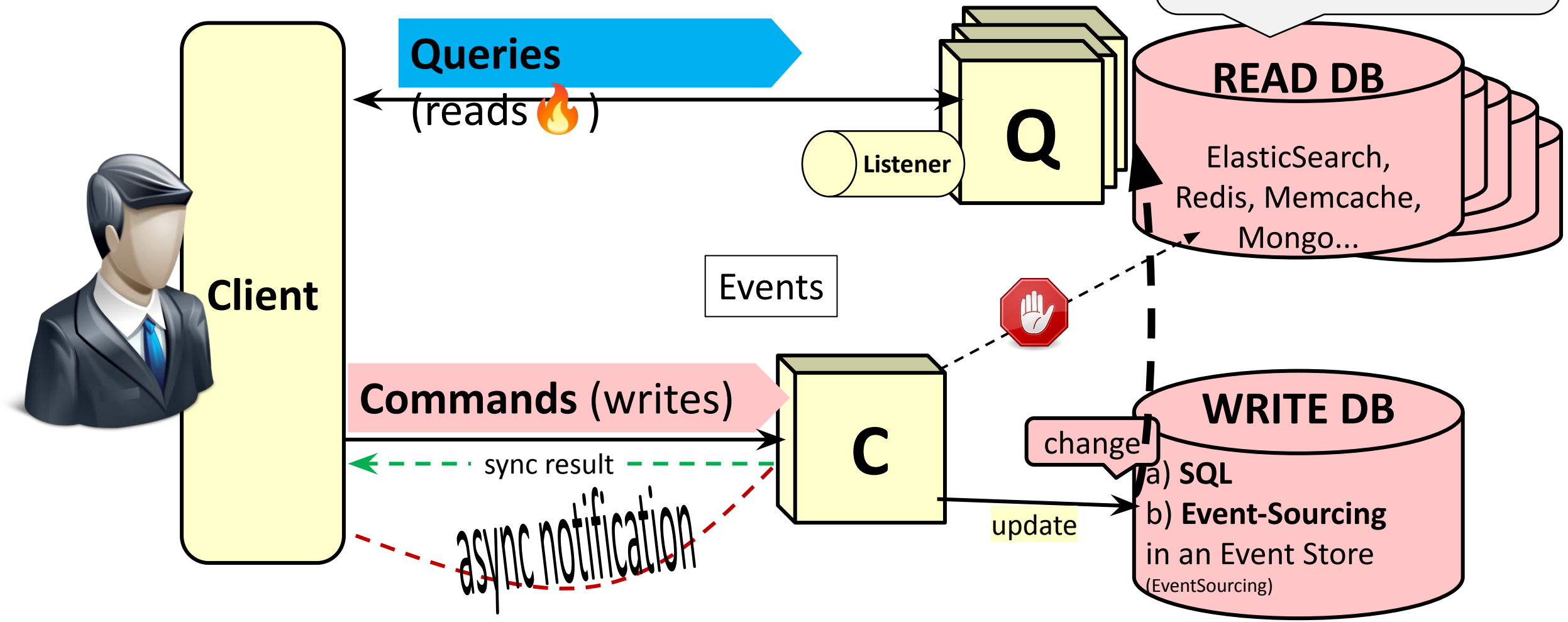
- **Redis**,... latency<1ms
- **Elastic**... full-text search
- **Mongo**... dynamic structure
- **Materialized Views** (ORA👴🏻) to pre-aggregate data regularly...

🔽 **Eventual** ⏱️ **Consistency**

low latency + high availability, under high load

Distributed CQRS

- Can store pre-built JSONs
- No need for DB constraints
- Horizontal scalable



= autonomous component

📺 CQRS by Greg Young, 2011 ⭐ https://youtu.be/JHGkaShoyNs

Distributed CQRS

... is more than SQL DB Read Replicas

CQRS = **optimize data for reading**: restructured, aggregated, enriched
DB Read Replicas = copy of data on another server

... is more than ETL (OLTP□OLAP for BI)

CQRS = **near-real-time** - event-streaming
ETL = runs at end-of-day/week/on-demand

⚠️ WARNING ⚠️

Many systems fit well with the notion of "Excel"
(=an information base that is updated in the same way that it's read).

Adding Distributed CQRS to such a system
can **add complexity**,
lower productivity, and
add **unwarranted risk**
to the project, even in the hands of a **capable** 🥷 **team**.

[Distributed] **CQRS is difficult to use we** Mainly due to async

~ Martin Fowler, ThoughtWorks

CQRS

Separate **Commands** (Updates)

from **Queries** (Reads)

Command **Query**

A POST or PUT should return data?

(besides server-generated **ID**)

vs your client call a GET after?

"enriched": polish 1 attr, add 2 more

...for "client convenience"

NO!*

- **Couples** GET and PUT responses
- **Privacy** issues if enriched data is sensitive
- **Performance Waste** if clients don't use that data

* ✅ Valid use: avoid "delay reading your own write" race in **Async CQRS**🤔

PUT should not return data

- Client/FE can follow up with a GET:

```
fetch("PUT", id…) // command (write)
+ .then(_ ⇒ fetch("GET", id...)) // query (read)
  .then(r ⇒ ...);
```

- **± Frontend #resistance => go FullStack**
- **± Server performance hit?** = +1 read for each write
  server waste = count (PUT)^{rare} x latency (GET)^{fast} = low <<< neglectible
- **± Client delay?** Is +50ms relevant for a human? 🤨
- **+ GET might return old data for a while after a PUT** 😱 in Async-CQRS, to *See my own changes*™
  Also:
  1) Study user impact ✊
  2) Reduce duration(inconsistency_window)

Debates>

"I Can't See My Own Changes"

User created/edited a record, but search results don't show it.

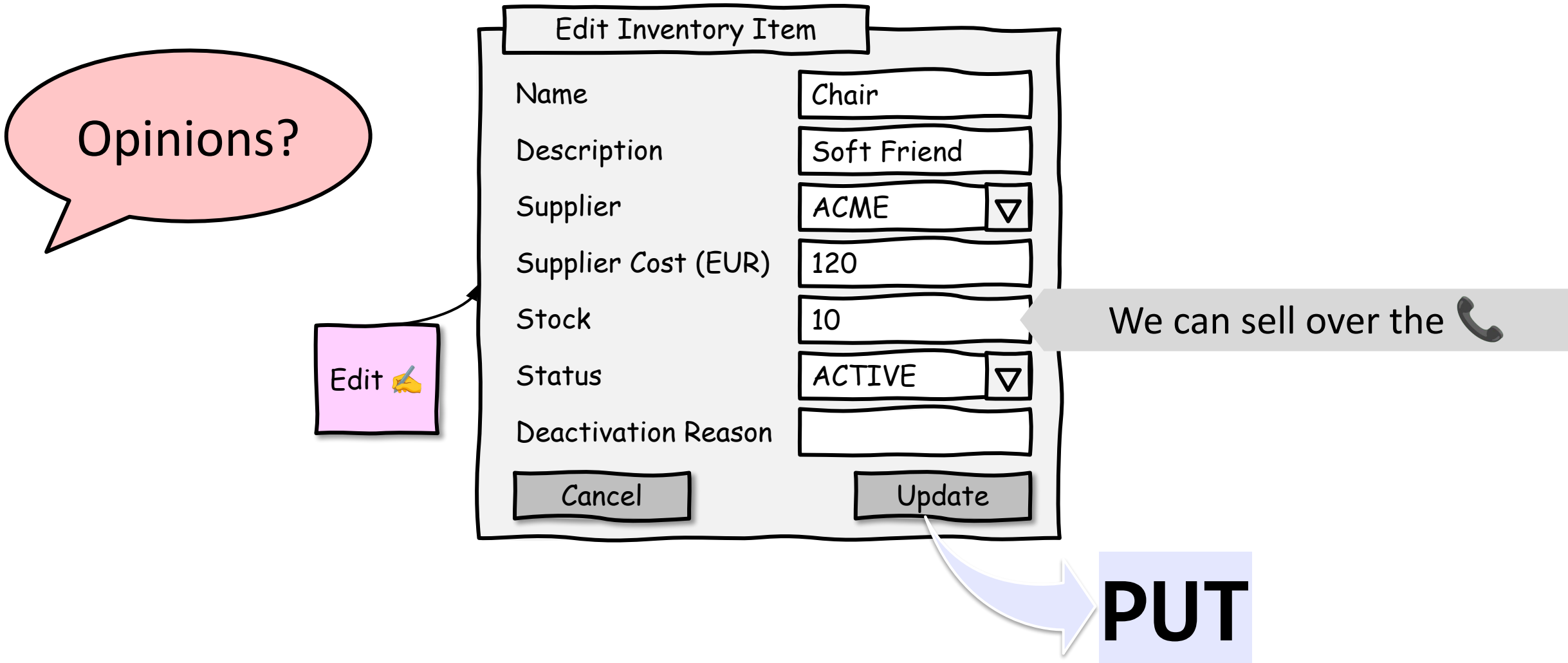
Approaches:

- a) NOT A BUG: convince the user to have patience 🧘
- b) Reduce chances by redesign UI (ie. trick the user)
  - ui-block +3sec after POST/PUT completes
  - change UI flow to NOT go back to SEARCH
- c)FE hack: display in DOM data returned from PUT/POST
- d)FE hack OMG:"Optimistic Update" ⛔❌
  - write in Browser DOM what FE hopes to come back from ES in 5sec 😱

Edit buttons!

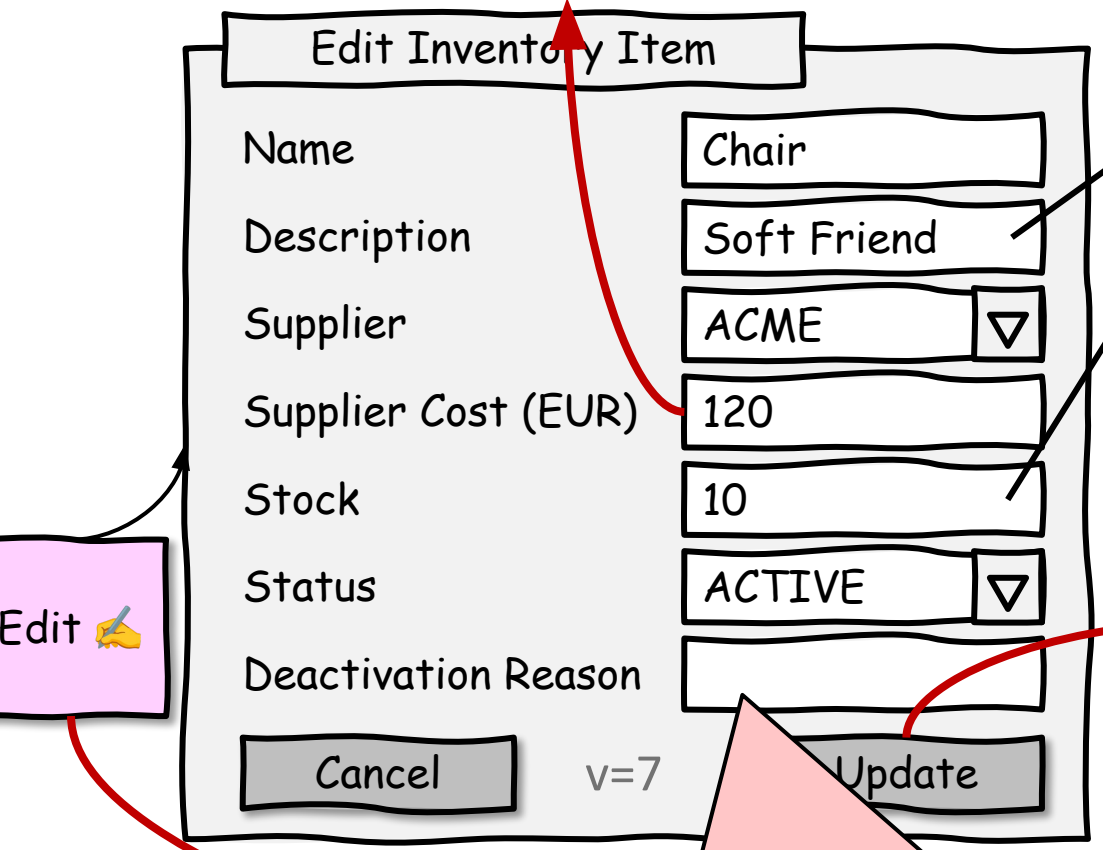
Don't you love them?

A Large Edit Screen



A Large Edit Screen

Server must DIFF new state vs DB



If you change status ACTIVE→INACTIVE, only then user must provide a reason

BAD UX: not obvious rule

CONCURRENT UPDATES

While a user edits the description for FOMO...

A customer buys an item in the web shop

To avoid **data loss** by blind overwriting stock=10:

A) **Send delta** = -3 (sold 3), not the **new total**

B) **optimistic locking:** (**can frustrate users**):

UNABLE TO SAVE

Someone else already changed this item.
Refresh the page and re-do your updates.

OK🤬

i) +VERSION column in DB - sent & received from clients

ii) +ETag request header matches hash (DB data)

C) **pessimistic locking** (**risk of bottleneck**):

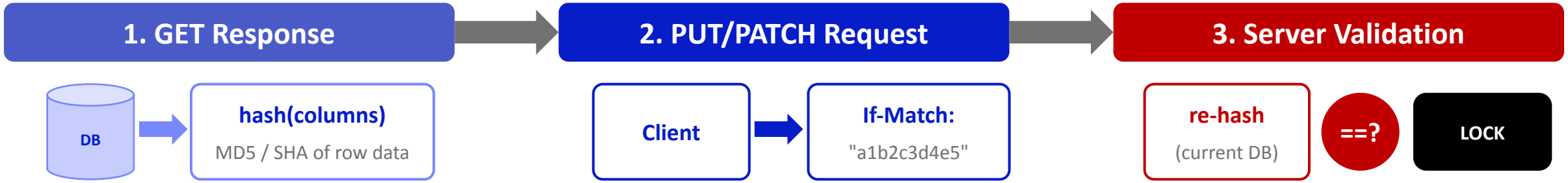
CANNOT OPEN EDIT SCREEN

Item under edit by **vrentea** since 3h ago.

OK🤬

+LOCKED_BY, LOCKED_AT columns in DB +lock expiration ⏱️

How an ETag Works



Server reads DB columns, computes a **hash** of the relevant columns, and returns it in the **ETag response header**

ETag: "a1b2c3d4e5"

Client sends back the **same ETag** in the **If-Match** request header along with the update payload

Server **re-hashes** current DB columns, compares with the If-Match ETag, wrapped in a **pessimistic lock**

Match = nobody changed it → **200 OK**, apply update

Mismatch = stale data → **412 Precondition Failed**

Pessimistic Lock Scope

The re-hash + comparison + DB write happen inside a **SELECT ... FOR UPDATE** (pessimistic lock) so no concurrent write can sneak between the check and the update

Task-Based UI

"Action Buttons over Edit Screens"

Name	Supplier	Active	Supplier Cost	Stock
Chair	ACME	☐	120	10
Armchair	ACME	☑	160	12
Table	ACME	☑	255	5
Sofa	ACME	☐	980	4

Edit Inventory Item

Name	Chair
Description	Soft Friend
Supplier	ACME
Supplier Cost (EUR)	120
Stock	10
Status	ACTIVE
Deactivation Reason	

Cancel | Update

Edit some text to increase FOMO

Adjust price & supplier

Sell over the phone

Deactivate a product

What actions my users ***usually*** do?

☢ #resistance of the expert 👴🏻 backoffice

Task-Based UI

✅ **Semantic-rich** API [+UI]
✅ Lower **concurrency** risk
✅ Simpler server implem.

❌ Requires **User Research** 💖
❌ **More Endpoints**
❌ **More UIs**

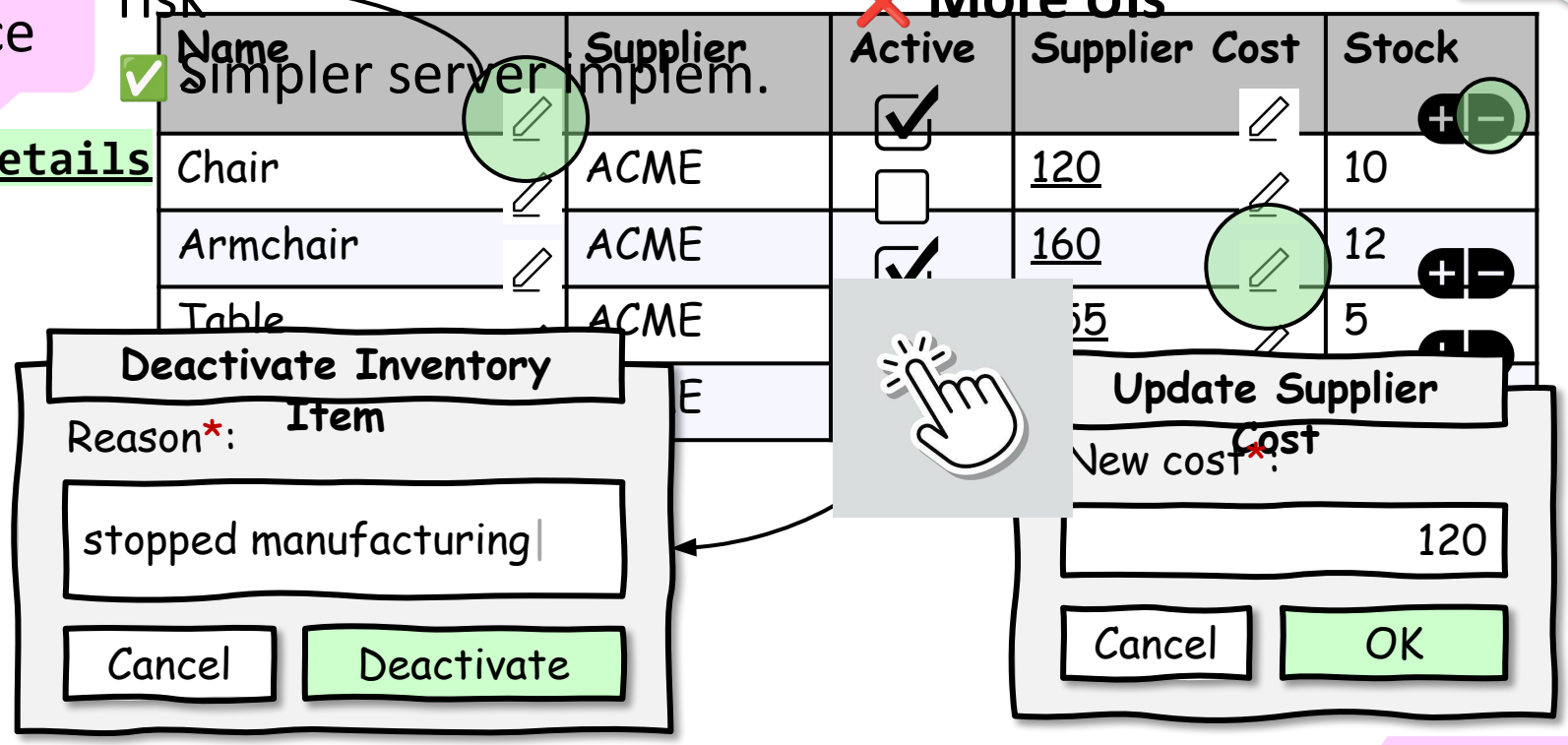
"imaginary" sub-resource ✅

```
PUT /items/13/details
{
 name:
 supplier:
 description:🦄
}
```

'POST' because action is **not idempotent**
=should not retry on timeout

```
POST /items/13/sell
{ quantity:2 }
```

verb! ✅




```
PUT /items/13/deactivate
{ reason: .. }
```

verb! ✅

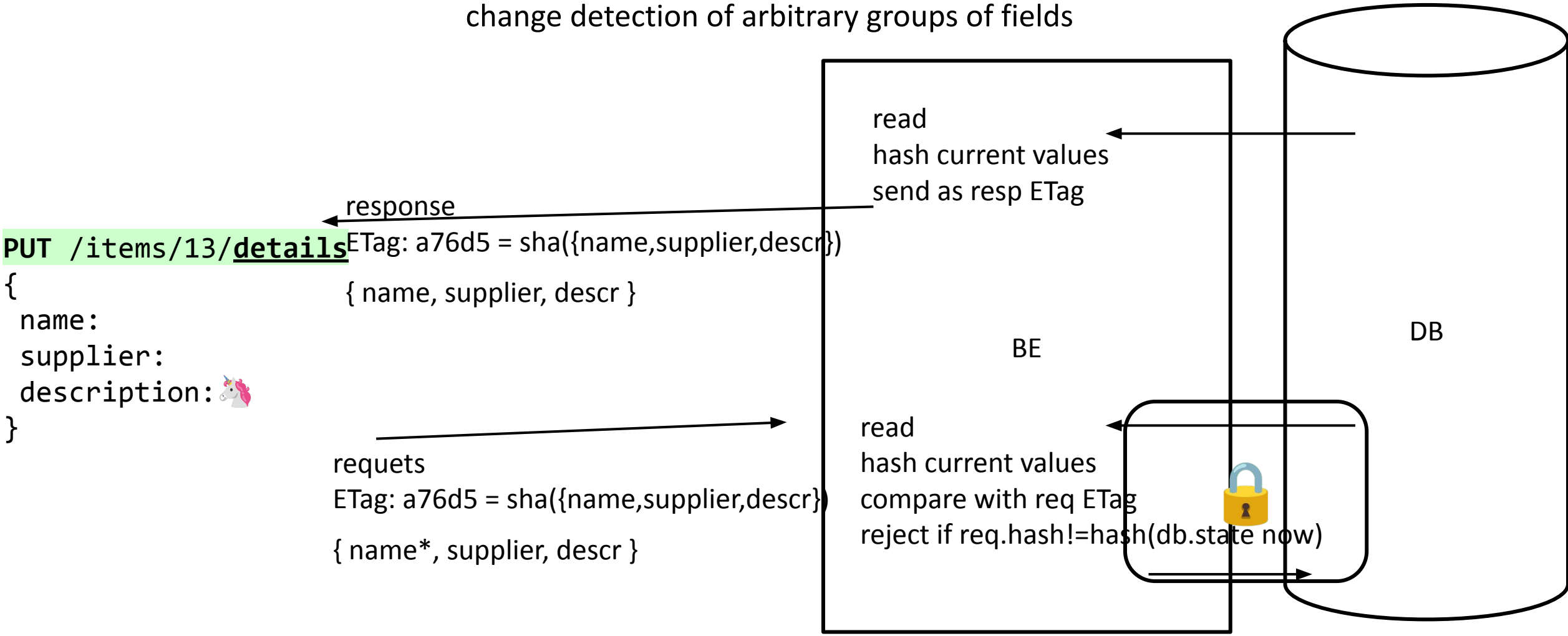
```
PUT /items/13/cost
{ newCost:120 }
```

sub-resource ✅

https://cqrs.wordpress.com/documents/task-based-ui/

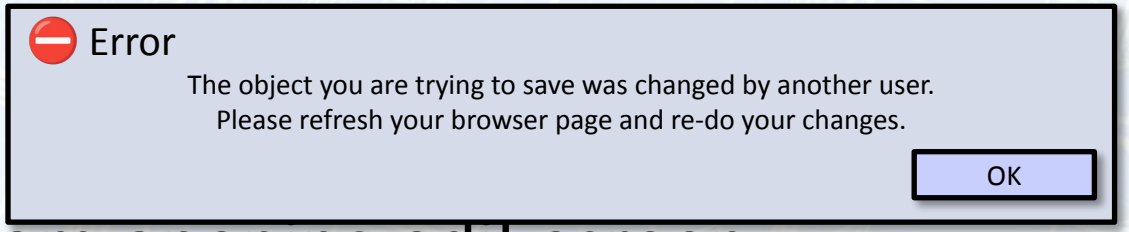
ETag

change detection of arbitrary groups of fields

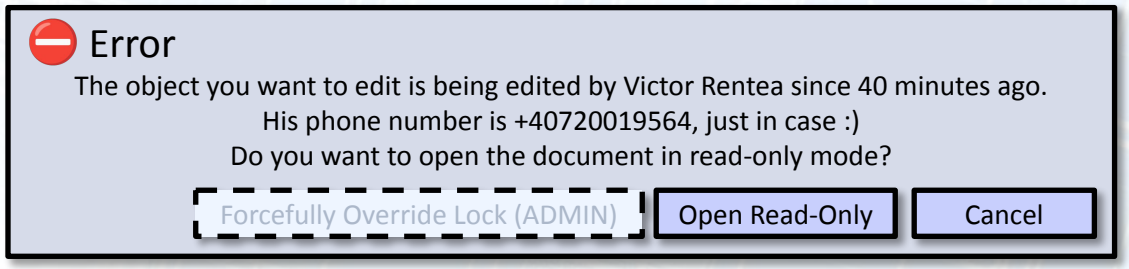


Concurrency Control

- **Optimistic Locking** = detect and reject the late change (2nd arriving)
  - On update, increment version in DB (@Version in ORM)
  - Before update, check that your copy is not outdated
  - User frustration: wasted effort



- **Pessimistic Locking** = prevent users from opening edit screen
  - Store the editor user in a “LOCK” table or in "IN_EDIT_BY_USER" column
  - Requires lock expiration
  - Careful release on any screen close:



Religious REST Fallacy

" URLs shalt never contain verbs "

PUT /items/13/**deactivate**
{ reason: .. }

verb ✅

💖

When noun URLs feel awkward, use verbs!

🤢 **POST** /items/13/**deactivation** + { reason: ..}

🤢 **DELETE** /items/13/**activation** + { reason: ..🤔}

🤢 **PUT** /items/13/**status**

```
{
  newStatus:"INACTIVE",
  deactivationReason:"<required only if newStatus=INACTIVE>"
}
```

Religious REST Fallacy

- When CRUD /<*noun*> limits your API semantics, introduce:
- **Sub-resources**: **PUT** /items/13/**cost** ~~**set-cost**~~
- **Actions** (verbs): **PUT** /items/13/**deactivate**
- **But avoid method-names:**
  - ❌ GET /get-item-by-name?q=
  - ❌ GET /items/**by-name**?q=
  - ❌ GET /items/**by-code**?q=
  - ❌ GET /items/by-name-and-code?**name**=..&code=..
  - ✅ GET /items?**name**=..&**code**=.. --- optional criteria
  - ✅ **POST** /items/**search** + {name:.., code:..} --- for larger/sensitive search forms

Separate unrelated client actions in different screens+endpoints

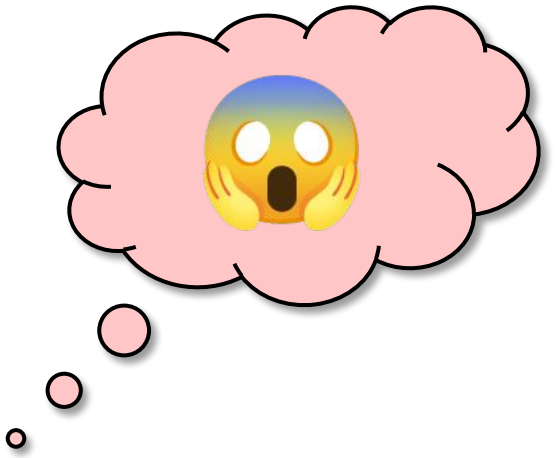
Segregate unrelated client actions

in different screens/endpoints

≈ a Git commit

changing 300 lines in 20 files

with message: "YOLO"



Lacks Intent

PATCH


partial update with
PATCH

PATCH

```
PATCH /items/1 = partial update
{
 "status": "INACTIVE",      => set
 "deactivationReason": "reason", => set
 "description": null        => remove
 // any other fields        => unchanged
}                => hard to parse
```

Why?
did **status**, **deactivationReason** and **description** change at once??

Client Intent?

JSON Path

jsonpatch.com

```
[
 {op:"set", path:"/status", value:"INACTIVE"},
 {op:"set", path:"/deactivationReason", value:"reason"},
 {op:"rem", path:"/description"}
 {op:"set", "path":"/authors[id='jdoe']/phone"}❌
]
```

PATCH

Lacks Semantics

```
var newEntity = mapper.fromDto(dto)

var oldEntity = repo.findById(id)

if (entityFromDB.status == ACTIVE &&
    entityFromDto.status == DEACTIVATED)
    // aka deactivation

   oldEntity.deactivationReason =
      requireNotNull(newEntity.deactivationReason)
```

PATCH

Lacks Semantics

```
PUT /items/1/deactivate
{reason:".."}
```

PATCH

Lacks Semantics

Valid use-case for PATCH:

The server doesn't enforce any rule about the received data.

"Pass-through data"

Tip: Store that data as json in SQL CLOB or in Mongo ☺️

Headers

Caching: ETag:<hash>, Last-Modified-Since: ignored by clients

Authorization: Bearer <jwt{userid (without roles}> requiring a call to role-service

Location: missing in POST response!

x-trace-id: missing from clients

Retry-After: ignored by clients, firing a higher request rate

DON'T HATEOAS with your own FE

Return 200 cu body: eroare tehnica cod 92746218

Internal Server Error

-added for Vivek from Mastercard, 1.5h before the talk at @DevoxxBE25

`{..}`

Security Vulnerability 🚨:
Hackers can determine frameworks versions and exploit their known vulnerabilities

`500 Internal Server Error`

```
java.lang.IllegalArgumentException: Missing email
at victor.training....CustomerApplicationService.register(CustomerApplicationService.java:77)
at java.base/java.lang.reflect.Method.invoke(Method.java:569)
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119)
at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
at java.base/java.lang.Thread.run(Thread.java:840)
```

WRONG

💭 Payload validation error

`errorRef:<UUID>`

`log👆 + errorRef`

`TraceId`

`400 Bad Request or 422`🤔

`Missing 'email'`

`{..,email}`

`Missing 'phone'`

`{..,email,phone}`

`Missing 'age'`

x 10 more times

`Missing ['phone','email','age'..]`

RFC 9457 = "standard" error response schema

RFC 9457/7807 = "standard" error response schema

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/problem+json
Content-Language: en

{
 "type": "https://example.net/validation-error",
 "title": "Your request is not valid.",
 "errors": [
            {
              "detail": "must be a positive integer",
              "pointer": "#/age"
            },
            {
              "detail": "must be 'green', 'red' or 'blue'",
              "pointer": "#/profile/color"
            }
          ]
}
```

JSON Path

https://www.rfc-editor.org/rfc/rfc9457.html#name-the-problem-details-json-ob

4xx <vs> 5xx

- **4xx Client Error** = Error in Payload; (400 or🤔 422,451..)
  - Payload validated against @ or a schema (OpenAPI/XSD..):
  - Required field missing @NotNull
  - Email has bad pattern @Email
  - Name too short @Size
  - Deserialization❌: xml instead of json, "42" instead of 42
  - Also: Retry a 4XX shouldn't help
  - 404 Not Found=∄ vs 403 Forbidden = ∃ but not for you
- **5xx Server Error** = Business Rule
  - Unique email in DB => 4XX or🤔 5XX✅ (as it requires a DB access)
    - Potential Security Breach as client finds out that email is NOT registered

Hint: in the excellent dry humor style of David Schmitz in '10 tips for failing at microservices' 🤣

REST API Design Pitfalls

- ✔Who needs Backwards Compatibility? Keep your clients agile!
- ✔Encourage your clients to call your GET {id} in a loop = traffic++
- ✔No rate limit! Just auto-scale:1000
- ✔Why should I hide my internal Domain Model? – KISS! to json!
- ✔Reuse the same DTO class in POST/PUT/GET! – DRY!
- ✔A PUT should return 'enriched' data back to client - #benice
- ✔CRUD is all you need! Burn any verbs from your URLs!

(Just to be sure🙈) DON'T DO the above

Thank

victorrentea.ro

It depends...

Almost everything I said has exceptions.

Approach me for debates.

🙏

Verbs in URL

```
paths:
  /getUser:
    get:
      summary: Get a user
      parameters:
        - name: id
          in: query
          schema:
            type: integer
  /createUser:
    post:
      summary: Create a user
  /deleteUser:
    post:
      summary: Delete a user
  /updateUser:
    post:
      summary: Update a user
```

Everything returns 200

```
paths:
  /users/{id}:
    get:
      summary: Get user by ID
      responses:
        '200':
          description: Returns user, or an error message if not found
          content:
            application/json:
              schema:
                type: object
                properties:
                  status:
                    type: string
                    example: "error"
                  message:
                    type: string
                    example: "User not found"
```

Inconsistent naming conventions

```
paths:
  /Users:
    get:
      summary: List all users
  /order_items:
    get:
      summary: List order items
  /GetInvoices:
    get:
      summary: List invoices
  /product-catalog:
    get:
      summary: List products
```

No pagination on collection endpoints

```
paths:
  /orders:
    get:
      summary: Returns ALL orders ever placed
      responses:
        '200':
          description: A list of every order
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Order'
```

Chatty API requiring many calls to do one thing

```
paths:
  /order/id:
    get:
      summary: Get order ID only
  /order/status:
    get:
      summary: Get order status only
  /order/customer:
    get:
      summary: Get customer ID on the order only
  /order/items:
    get:
      summary: Get line items on the order only
  /order/total:
    get:
      summary: Get order total only
```

Mutating state with GET

```
paths:
  /invoice/send?id=88:
    get:
      summary: Sends invoice #88 to the customer and marks it as sent
  /user/deactivate?id=12:
    get:
      summary: Deactivates user account #12
  /cart/checkout?id=5:
    get:
      summary: Processes payment and empties cart
```

Fun Times in the Industry

- ANAF API returns 200 + {failed: true}
- System returns 500 + "Timed Out" but request completed OK
- On a failure, my clients retry 5 times without checking if any call was ok: for (i=1..5) call
- I offer a GET /many + [1,2,3..] but clients call me for () api.get([list[i]]); // 1 element/list
- I 429 rate limit my clients, but clients raise ticket that I fail too often. They win.
- If we don't find a record in a 3rd party API we return 503. Clients retry.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<header xmlns="mfp:anaf:dgti:efactura:mesajEroriFactuta:v1" Index_incarcare="123" Cif_emitent="123">
    <Error errorMessage="CUI-ul identificat in factura=123 este radiat"/>
</header>
```