TOP

REST API

DESIGN PITFALLS

BY VICTOR RENTEA

- Versioning
- Performance Tricks
- Growing Complexity
- CQRS Levels
- Semantic APIs
- Error Handling

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** +=

REST API

Co-author of **TCP/IP**

Postel's Law

Be conservative in what you do,

but liberal in what you accept from others

Backwards Compatibility

How to Break It?

Cause a Breaking Change!

that would force clients to update

Example: ± a field?

a request

POST /users/ {email}

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

fullName:{   *required
2   "firstName": "John",
3   "lastName": "DOE",
4   "emailAddress""a/com",
5   "phone": ["+407129"],
6  -~~"currency"~~:  "EUR"
}  +"address" *required
   +"middleName" ⭐ optional
```

array

+regex pattern/min length

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

One of ~~RUB~~|EUR|GBP

no longer supported

(intentional?)

a response

"2023-03-01T00:00:00Z"

"2023-03-01" (ISO)

```
{
7 "date": "03/01/2023",
8  age": "37"←number
9 -~~"country"~~: "ROU"
  +"extra": ⭐
}
```

One of ~~ROU~~|ESP|NOR

became supported

Legend

breaking change

backwards-compatible change

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, Traceparent💖
- **Convince v1 Clients to Upgrade**
  - 📞, 🙏, 🤬, **No New Features**, Rate Limit↘, Fees↗; force mobile app update😫
  - Offer help: Guides, Automatic OpenRewrite recipes,... Can I code with you?
  - **Don't**: ~~if (random) throw "Upgrade!"~~ or ~~sleep(1s)~~ or ~~claim 'V1 is vulnerable'!~~
- **Ideally: support <u>max 2</u> parallel versions for a <u>limited time</u>**🤞


?!
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,?>
}
```

in case **tomorrow** we start supporting more 😊

(but today you require/return size=1)

more *types* **tomorrow**?

more keys **tomorrow**?

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

Versioning Strategies

- **new microservice**: order-service-**v2** + separate Git + CI ± **DB**😱
  - **Design for deletion** – easy to remove v1 later (soon!)
- **per-service**⭐: order-service.intra/**v2**/order/{id}
  - "/v1/" is a middle finger🖕to your API clients, REST author Roy Fielding
- **per-endpoint**: /order/**v2**/{id} --- multiple bounded contexts?
- **Content-Type + Accept**: application/json+**v3**

Contract Tests

Auto-detect contract mismatch

#musthave

Generate FE Types from BE API

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

if client=Browser

-v2.0.jar 🤔

api-client-v1.1.jar

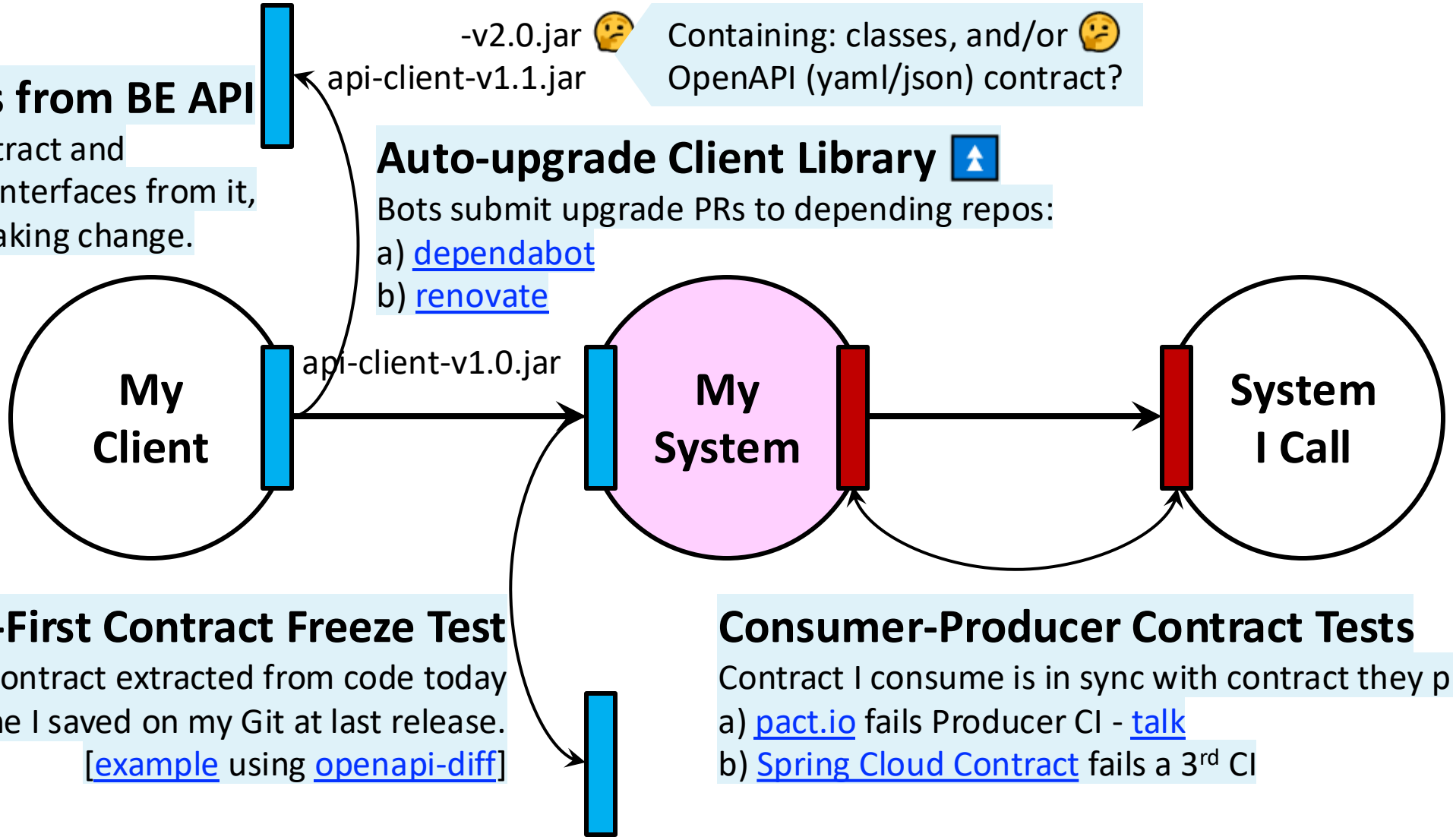
Containing: classes, and/or 🤔 OpenAPI (yaml/json) contract?

Auto-upgrade Client Library ⏫

Bots submit upgrade PRs to depending repos:

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

Performance

Get One of Many

What can go wrong?

You have many records in your DB, exposed via:

`GET /products/{id}`

Misplaced Responsibility?

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,...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 it ⚠️
- 😉 Rate-limit client calls : 429 Too Many Requests + 'Please use the batch API'
- On

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?

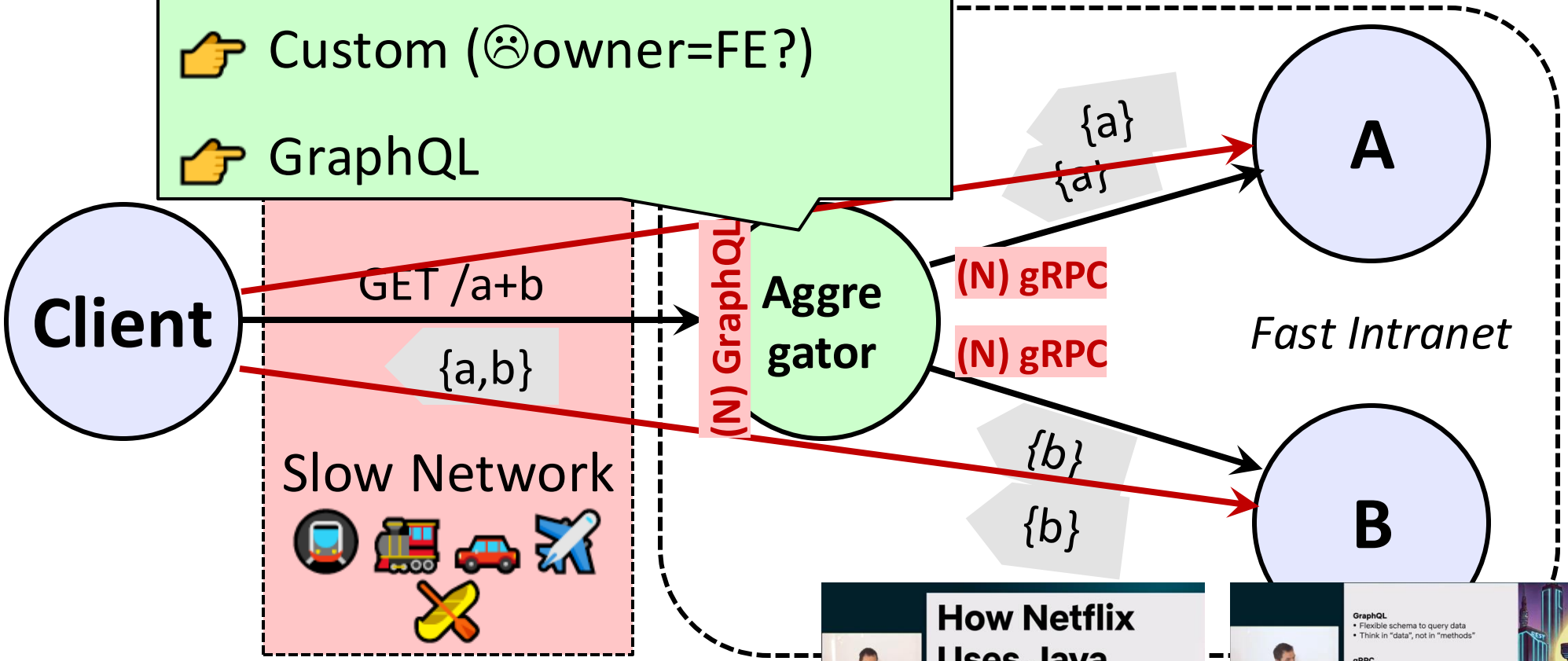
⏱️ Timeout: How long would this call take?

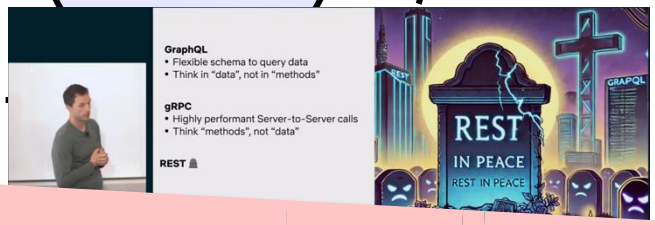
Aggregator

☺ **Performance & Decoupling**

👉 Custom (☹owner=FE?)

👉 GraphQL





Kind Reminder: you are NOT Netflix

Apr 25, 2025 @ JavaOne - 165k views

(new features)

Performance

Complexity

Scalability

Don't Expose Internal Model in your API

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

CustomerDto

Opinions?

❄️ **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

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

  ...
  String name;

  @JsonIgnore
  String phone;

  @JsonFormat(..)
  LocalDate birthDate;

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

stable clear documented

Separate **Contract** (API DTO)

from **Implementation** (Domain Model)

Evolving to Simplify Logic

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 Segregation

Update Data

Read Data

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..

🏆 **latency & availability**

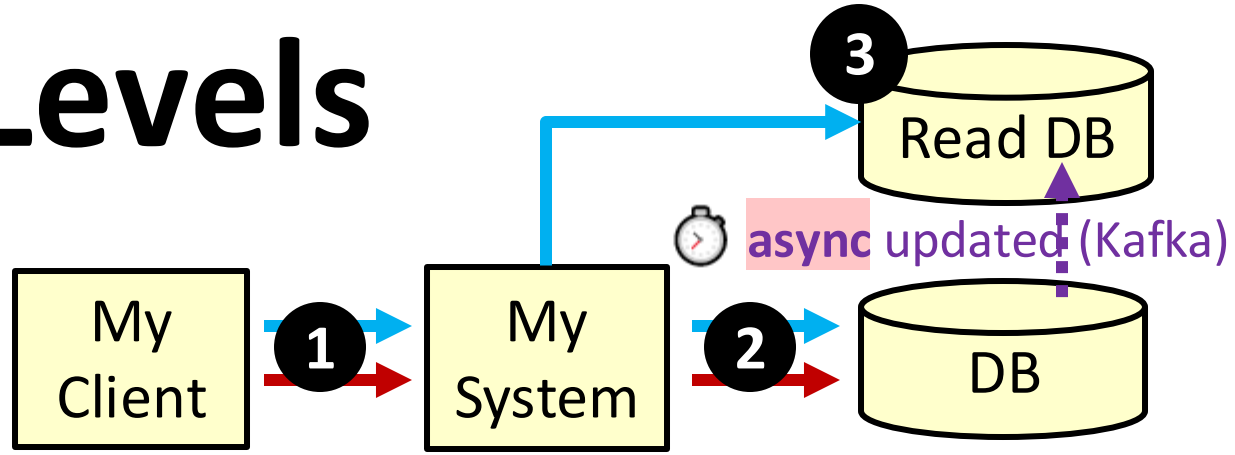
WRITE stores additional metadata: createdBy=, ...

🏆 **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 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

CQRS

Separate **Commands** (Updates)

from **Queries** (Reads)

Command

Query

A POST or PUT should return data?

(besides server-generated **ID**)

"enriched"

...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**

Edit buttons!

Don't you love them?

A Large Edit Screen

Opinions?

Edit ✍

Edit Inventory Item

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

Cancel | Update

We can sell over the 📞

PUT

A Large Edit Screen

Server must DIFF new state vs DB

Edit Inventory Item

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

Cancel v=7 Update

Edit ✍

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

This can cause **data loss** by blind overwriting stock=10.

Solutions:

9

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

+VERSION column in DB sent & received from clients

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 + watchdog

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 do my users 😍 ***usually*** do?

Task-Based UI

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

- ❌ Requires **User Research** 💖
- ❌ **More APIs**
- ❌ **More UIs**

sub-resource ✅

```
PUT /item/13/details
{
 name:
 supplier:
 description:
}
```

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

Deactivate Inventory Item

Reason*:

stopped manufacturing

Cancel | Deactivate

Update Supplier Cost

New cost*:

120

Cancel | OK

verb ✅

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

'POST' because action is not idempotent (not retryable)

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

verb ✅

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

sub-resource ✅

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

Religious REST Fallacy

" URLs shalt never contain verbs "

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

verb ✅

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

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

🤢 **PUT** /item/13/**status**

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

Awkward 🤨

Religious REST Fallacy

- When CRUD /<*noun*> limits your API semantics, introduce:
- **Sub-resources**: **PUT** /item/13/**cost** ~~**set-cost**~~
- **Actions** (verbs): **PUT** /item/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


PATCH

PATCH

```
PATCH /item/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

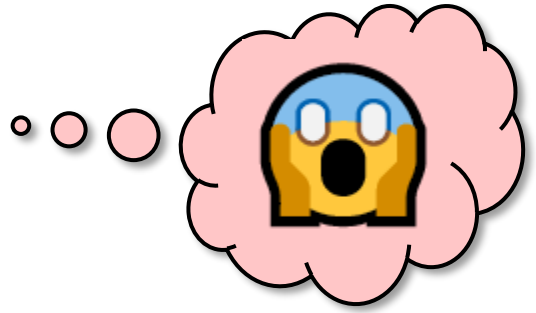
```
[
 {op:"set", path:"/status", value:"INACTIVE"},
 {op:"set", path:"/deactivationReason", value:"reason"},
 {op:"rem", path:"/description"}
 ~~{op:"set", "path":"/authors[id='jdoe']/phone"}~~ ❌
]
```

PATCH

Lacks Semantics

≈ a Git commit of 300 lines in 20 files

having the message: "did good"



PATCH

Lacks Semantics

```
newEntity = mapper.fromDto(dto)
oldEntity = repo.find(id)

if (oldEntity.status == ACTIVE &&
    newEntity.status == DEACTIVATED)

   newEntity.deactivationReason = reason
}
```

PATCH

Lacks Semantics

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

PATCH

Lacks Semantics

Valid use-case for PATCH:

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

"Pass-through data"

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

500

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

errorRef:<UUID>

& log exception with errorRef

💭 Payload validation error

400 Bad Request or

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 = "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

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?** Clients should be agile 💪
- ✓ **Encourage your clients to call your GET {id} in a loop = traffic++**
- ✓ **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! #PUT**
- ✓ **One PATCH to rule them all! Mystery time: why?**
- ✓ **Errors don't happen to you! (nor to Chuck Norris)**

(Just to be sure🙈) DON'T DO the above stuff👆

Thank You!

victorrentea.ro

It depends...

Almost everything I said has exceptions.

Approach me for debates.