

The 7 layers of a predictable API

An API that never surprises its client is the consistency of seven small decisions.

Muhammet Şafak • October 3, 2026



Four stops

01 Resource and status code Layers 1 and 2

02 List endpoints Layer 3: pagination, filtering, sorting

03 Response and error Layers 4 and 5

04 Version and contract Layers 6 and 7



Every layer is a consistency decision

Resource

The URL is noun-based and plural; the HTTP method carries the action.

Status

Success is 2xx, client error is 4xx, server error is 5xx.

List

page, per_page, sort, order: the same names on every endpoint.

Envelope

Every response has the same skeleton: success, data, message, errors.

Error

Errors come back as error.code, error.message, error.details.

Version

A breaking change calls for a new version.

OpenAPI

The contract is written before the code.

The method carries the action, not the URL

METHOD	MEANING	EXAMPLE
GET	Read	GET /users/42
POST	Create	POST /users
PUT	Replace entirely	PUT /users/42
PATCH	Update partially	PATCH /users/42
DELETE	Delete	DELETE /users/42

The status code tells the outcome

CLASS	CODES	MEANING
2xx	200 • 201 • 204	Succeeded; created; no content
4xx	400 • 401 • 403 • 404 • 422	Bad data, identity, permission, not found, validation
5xx	500	Unexpected error on the server side

The identity of a page

2

current_page

last_page: 15

20

per_page

Upper limit: min(\$adet, 100)

21–40

from – to

Records on this page

287

total

Total records

EXAMPLE FROM THE SOURCE • OFFSET

Deeper pages, costlier OFFSET.

To reach page ten thousand, the database has to read through and skip 200,000 records; one solution is cursor pagination.

Every response has the same skeleton

The client reads the same four fields on every endpoint: `success`, `data`, `message`, `errors`.

```
● ● ● response.json

1 {
2   "success": true,
3   "data": {},
4   "message": null,
5   "errors": null
6 }
```

Lists use the same envelope

A paginated list comes back as `data.items` and `data.meta`; the envelope does not change.

response.json

```
1 {
2   "success": true,
3   "data": {
4     "items": [],
5     "meta": {
6       "current_page": 1,
7       "per_page": 15,
8       "total": 243,
9       "last_page": 17
10    }
11  },
12  "message": null,
13  "errors": null
14 }
```

An error is a contract too

`code` is fixed for machines, `message` can change for humans, `details` carries extra context.

error.json

```
1 {
2   "error": {
3     "code": "VALIDATION_FAILED",
4     "message": "Gönderilen veriler doğrulanamadı.",
5     "details": [
6       {
7         "field": "email",
8         "message": "Geçerli bir e-posta adresi giriniz."
9       },
10      {
11        "field": "phone",
12        "message": "Telefon numarası zorunludur."
13      }
14    ]
15  }
16 }
```

Envelope and error rules



ERROR WITH 200

Returning 200 + "error": true for errors works but goes against REST; caching and logging break.



FORMAT

Error codes in uppercase with underscores: QUOTA_EXCEEDED.



CODES

New codes can be added, old codes should not be removed.



SCALE

In a small service with a single client, a status code and a short message are enough.

Three strategies, URI preferred

STRATEGY	EXAMPLE	TRADE-OFF
URI	<code>GET /api/v1/users</code>	Clearly visible in logs and proxies; multiplies URLs
Header	<code>X-API-Version: 2</code>	URLs stay clean; a missing header creates ambiguity
Query	<code>GET /api/users?version=2</code>	Practical at the start; proxies and CDNs may not see it

At a basic level: which change calls for a new version?

✓ Compatible

Adding a new field to the response.

Adding an optional parameter.

Enriching error messages.

✗ Breaking

Removing or renaming an existing field.

Adding a required parameter.

Changing the response structure or the status code.

DEPRECATION

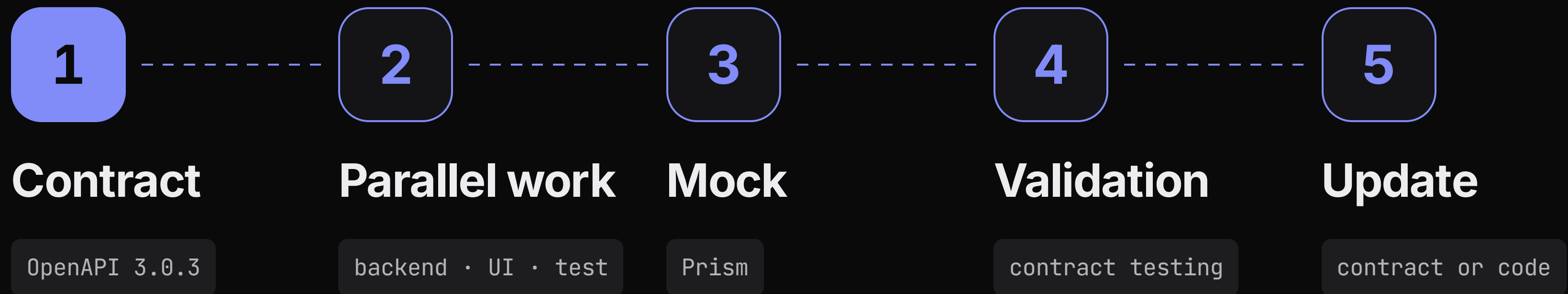
Give at least 6–12 months' notice.

For an API open to external clients, a deprecation period of at least 6-12 months is reasonable.

Announce a version's lifespan with a header

HEADER	DOCUMENT	WHAT IT SAYS
Deprecation: @1780272000	RFC 9745	When it was deprecated; the value is a date, true is invalid
Sunset: Sat, 31 Oct 2026 23:59:59 GMT	RFC 8594	When it will stop responding
Link: <.../docs/migrate-v2>; rel="deprecation"	RFC 9745	The address of the migration guide

Contract first, code later



The contract is written before the code

The schema excerpt of the example contract. The `/orders` endpoint restricts the `status` filter to `pending`, `completed`, `cancelled`.

```
openapi.yaml

1 components:
2 schemas:
3   Order:
4     type: object
5     required: [id, status, total]
6     properties:
7       id:
8         type: integer
9       status:
10        type: string
11       total:
12        type: number
```

When does contract-first pay off?



SMALL
PROJECT

One developer, small project: contract-first can be overkill.



MANY
CONSUMERS

Web, mobile, third parties: it definitely saves time.



LONG LIFE

In an API that will live for years, the contract is a decision record.

Thank you

muhammetsafak.com.tr

Seven layers, seven promises of consistency. The details are in the six source posts.

