M04-02 · GIBALJA VISUAL MANUAL

API 명세서 읽고 쓰기

> **한 문장 목표:** endpoint 하나가 누구에게 어떤 입력을 받아 무엇을 보장하고 어떤 오류를 돌려주는지 사람·도구·테스트가 함께 읽을 수 있는 계약으로 작성합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	endpoint 카드, OpenAPI 초안, 검수 기록

"`POST /orders`로 주문 정보를 보내면 성공합니다"는 호출 힌트입니다. "OAuth2의 `write:orders` scope가 필요하고, JSON body의 `productId`와 `quantity`가 필수이며, 성공하면 `201`·`Location`·Order resource를 반환합니다. 입력·인증·권한·재고·일시 장애는 `400·401·403·409·503` problem details로 구분합니다"까지 닫아야 계약입니다.

그림 1. endpoint 한 개를 여덟 칸으로 읽습니다. 주소·요청·성공만 보지 않고 권한·알려진 오류·실행 가능한 예시까지 확인합니다.

1

문맥

version·server

2

대상

resource·path

5

권한

security·scope

6

성공

status·header·body

좋은 명세 = 사람이 의도를 이해하고 도구가 구

3

행동

method·operationId

4

입력

parameter·body

7

오류

known error·problem

8

예시·재사용

examples·$ref

구조를 읽으며 테스트가 같은 결과를 판정한다

1. 이 PDF를 공부하는 방법

1회차 · 여덟 칸 읽기 · 15분

`문맥 → 대상 → 행동 → 입력 → 권한 → 성공 → 오류 → 예시·재사용` 순서를 익힙니다. 기존 OpenAPI 문서를 열면 이 순서로 한 operation만 먼저 읽습니다.

2회차 · 세 쌍 구분하기 · 30분

`server와 path`, `parameter와 request body`, `schema와 example`을 구분합니다. `openapi` version과 `info.version`, body 자체의 `required`와 field 목록의 `required`도 함께 비교합니다.

3회차 · 여덟 결함 실행하기 · 35분

API 명세 계약 스튜디오에서 완전한 계약과 일곱 결함을 검수합니다. JSON을 직접 바꾸고 점수·진단·endpoint 카드가 어떻게 달라지는지 봅니다.

4회차 · 실제 endpoint 작성하기 · 40분

API endpoint 명세 템플릿에 교육 신청·문의 등록·문서 승인 중 하나를 적고 OpenAPI JSON 초안을 만듭니다.

5회차 · 셀프 테스트 · 15분

정답을 가리고 12문제를 풉니다. 틀린 문제는 `위치`, `필수`, `완료`, `권한`, `오류`, `예시` 중 하나로 분류합니다.

학습 완료 기준

"API 문서가 부족합니다"를 " `POST /shops/{shopId}/orders`의 `shopId`가 path template에는 있지만 `in:path, required:true` parameter가 없고, 201에는 생성 resource의 Location이나 ID가 없으며, 보호된 operation인데 401·403과 security scope가 없습니다"처럼 위치와 계약 요소로 설명할 수 있습니다.

2. OpenAPI Description은 API의 표면과 의미를 기술합니다

OpenAPI Specification(OAS)은 HTTP API의 표면과 의미를 기계가 읽을 수 있는 형태로 기술합니다. 문서는 한 JSON·YAML 파일일 수도 있고 여러 문서가 reference로 연결될 수도 있습니다.

```
API 구현: 실제 요청을 받고 판정·저장·응답하는 시스템
API 명세: 그 표면·입력·결과·보안 조건을 기술한 계약
```

명세가 있다고 구현이 자동으로 안전해지는 것은 아닙니다. 반대로 구현이 동작한다고 소비자가 계약을 알 수 있는 것도 아닙니다. 문서와 runtime을 contract test로 비교해야 합니다.

2.1. 두 version을 구분합니다

```
{
  "openapi": "3.2.0",
  "info": {
    "title": "Order API",
    "version": "1.0.0"
  }
}
```

field	뜻	질문
`openapi`	문서가 따르는 OAS version	validator·문서 도구가 3.2.0을 지원하는가
`info.version`	API description·서비스 계약의 version	어떤 변경판인지 팀이 어떻게 관리하는가

`openapi: 3.2.0`이라고 API 제품이 자동으로 3.2가 되는 것은 아닙니다. 또한 최신 명세를 쓴다는 이유만으로 기존 도구 호환성을 가정하지 않습니다.

2.2. root에서 먼저 볼 것

field	역할
`openapi`	OAS version
`info`	title·version·설명

field	역할
`servers`	호출 기준 URL과 환경
`paths`	path와 operation
`components`	재사용 가능한 schema·response·security scheme 등
`security`	API 전체 기본 보안 요구
`webhooks`	API provider가 시작할 수 있는 수신 요청 기술

OAS 3.2.0 root에는 `components`, `paths`, `webhooks` 중 적어도 하나가 있어야 합니다.

3. 읽기 순서를 여덟 칸으로 고정합니다

순서	찾을 위치	읽을 질문
1	`openapi·info·servers`	어느 문서·환경인가
2	`paths` key	어떤 resource 관계인가
3	method· `operationId`	어떤 행동·업무 결과인가
4	`parameters·requestBody`	어디서 무엇을 받는가
5	`security`	어떤 scheme·scope가 필요한가
6	`responses` 2xx	무엇까지 성공인가
7	`responses` 4xx·5xx	어떤 알려진 실패가 있는가
8	`schema·examples·$ref`	실행·재사용 가능한가

30초 확인

문서를 처음부터 끝까지 읽지 않습니다. 사용자 흐름과 연결된 method·path 하나를 고르고 여덟 칸만 닫습니다. 한 operation을 제대로 읽은 뒤 공통 component로 넓힙니다.

4. server·path·method로 endpoint를 찾습니다

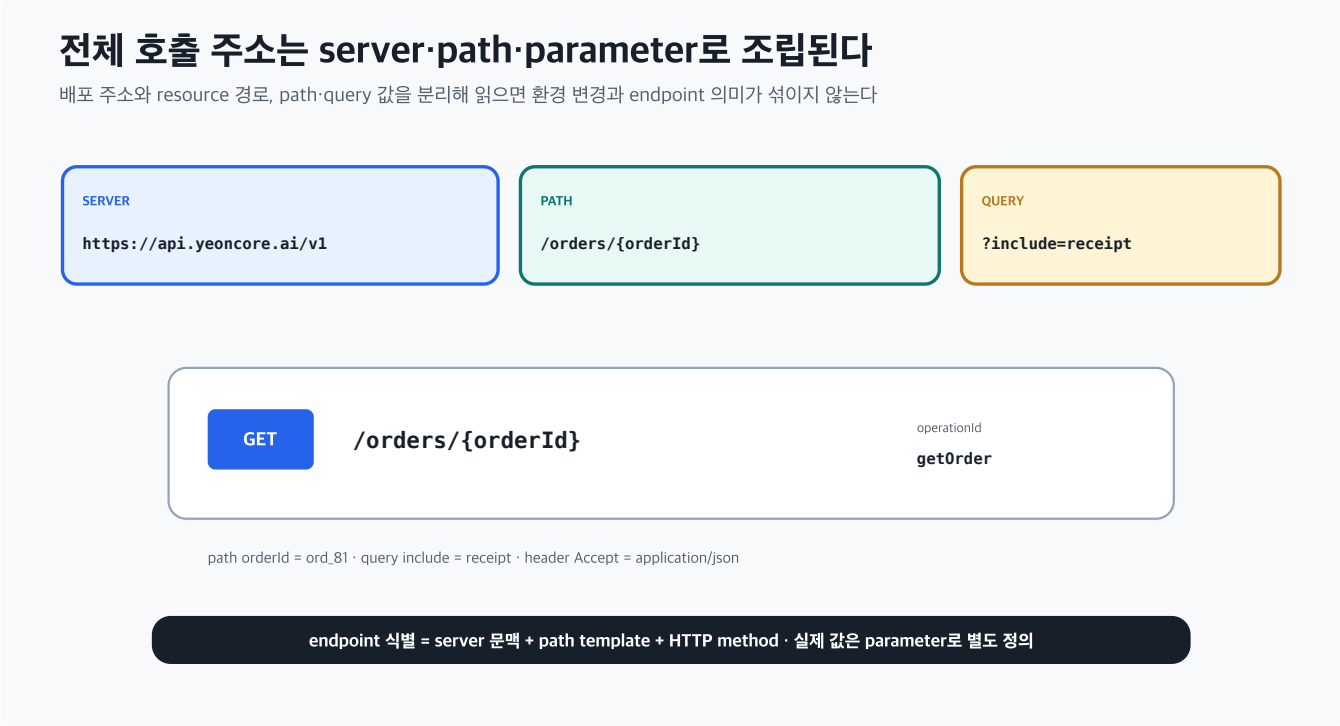


그림 2. 실제 호출 주소는 server URL에 path를 붙이고 template·query 값을 채워 만듭니다. operation은 path와 method의 조합입니다.

SERVER

https://api.yeoncore.ai/v1

PATH

/orders/{order

GET /orders/{orderId}

path orderId = ord_81 · query include = receipt · header Accept = applic

endpoint 식별 = server 문맥 + path template +

Id}

QUERY

?include=receipt

operationId

getOrder

cation/json

HTTP method · 실제 값은 parameter로 별도 정의

4.1. server는 배포 문맥입니다

```
"servers": [
  {"url": "https://api.yeoncore.ai/v1", "description": "production"},
  {"url": "https://sandbox-api.yeoncore.ai/v1", "description": "sandbox"}
]
```

환경이 달라도 resource path의 의미는 유지됩니다. 실제 secret·내부 host를 문서 예시에 넣지 않습니다.

4.2. path는 resource 관계를 드러냅니다

```
/shops/{shopId}/orders
/orders/{orderId}
/orders/{orderId}/receipt
```

`/createOrderNow`처럼 행동을 path에 반복하기보다 resource를 path에 두고 method와 `operationId`로 행동을 말하면 읽기가 쉽습니다. 절대 규칙이라기보다 일관된 설계 원칙이며, 업무상 command endpoint가 필요하면 효과·중복·상태를 구체적으로 문서화합니다.

4.3. method에는 HTTP 의미가 있습니다

그림 3. 같은 resource라도 method가 달라지면 요청 body·성공 status·안전성·멱등성·권한 계약이 달라집니다.

GET

목록·한 건 조회

safe·idempotent

`/orders/{id}`

상태·body·error 별도

POST

새 주문 생성·처리 요청

효과 계약 확인

`/orders/{id}`

상태·body·error 별도

PUT

resource

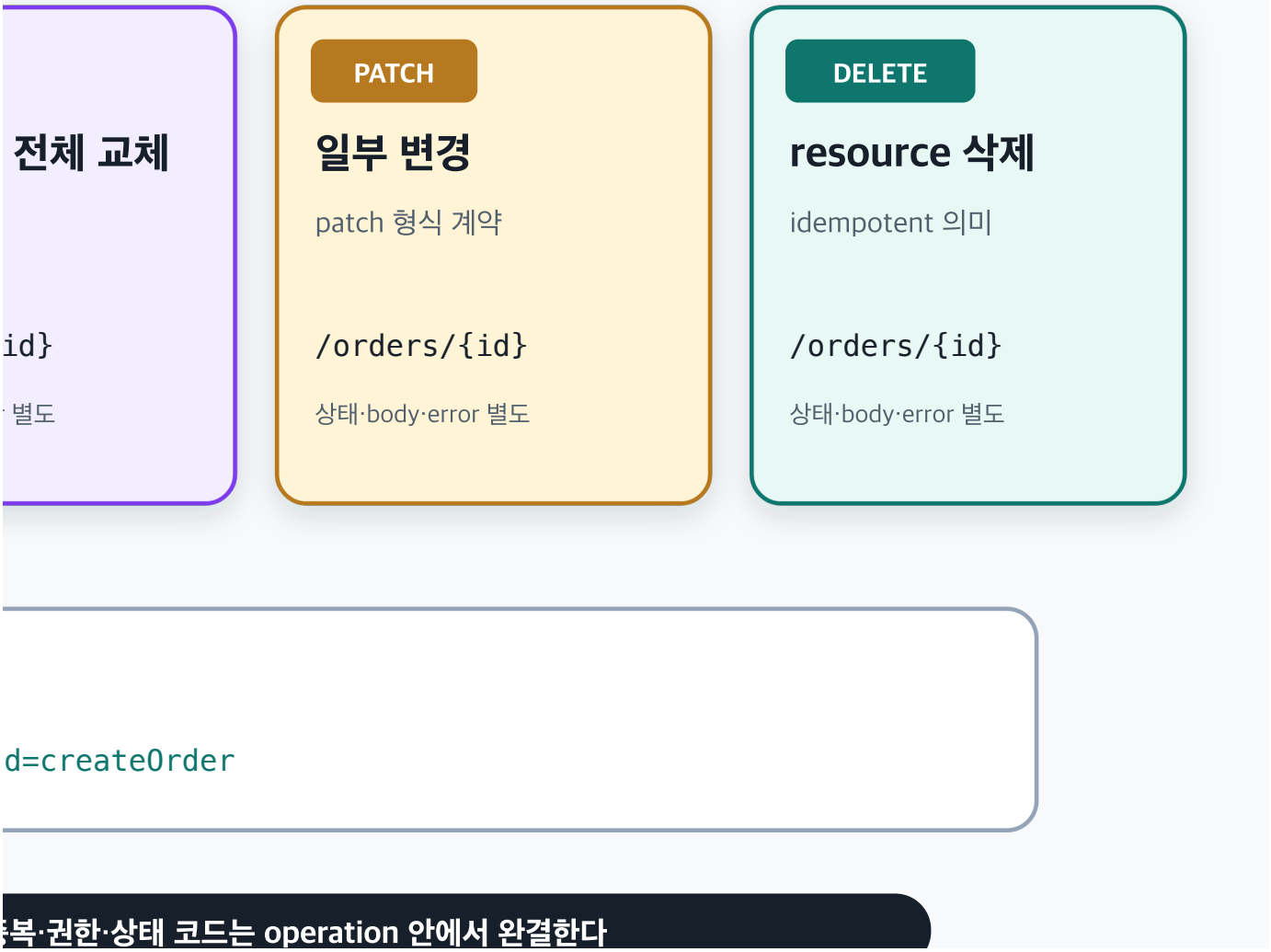
idempotent

`/orders/{`

상태·body·error

피해야 할 path	`POST /createOrderNow`
더 읽기 쉬운 계약	`POST /orders · operationI`

HTTP method 의미는 출발점이다 · 업무 효과·중


전체 교체
id}
별도
PATCH
일부 변경
patch 형식 계약
/orders/{id}
상태·body·error 별도
DELETE
resource 삭제
idempotent 의미
/orders/{id}
상태·body·error 별도
d=createOrder
복·권한·상태 코드는 operation 안에서 완결한다

method	대표 의도	확인할 것
GET	resource 조회	상태 변경을 요구하지 않는가
POST	생성·처리 요청	중복 효과와 생성 결과
PUT	target resource 전체 교체	전체 표현·멱등 의미
PATCH	부분 변경	patch media type·변경 규칙
DELETE	target resource 삭제	존재하지 않을 때·되돌림·권한

RFC 9110의 method 의미는 출발점입니다. “POST니까 무조건 201”, “DELETE니까 body 없음”처럼 외우지 말고 실제 operation 결과와 status 계약을 봅니다.

4.4. operationId는 도구가 쓰는 고유 이름입니다

OAS 3.2.0은 `operationId`가 API 안의 모든 operation 사이에서 고유해야 한다고 규정합니다. 대소문자를 구분하며 client SDK·link·test가 참조할 수 있으므로 일관된 naming 규칙을 정합니다.

```
getOrder
listOrders
createOrder
cancelOrder
```

5. parameter 위치를 먼저 구분합니다

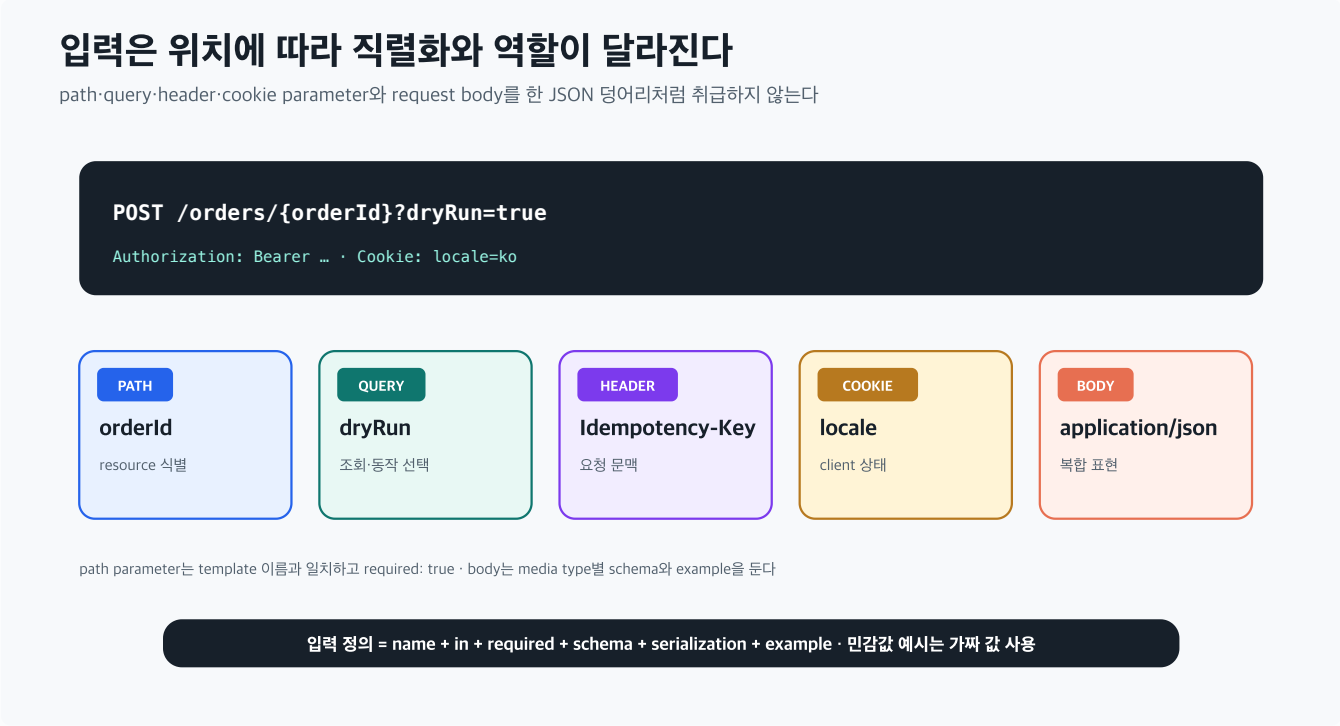


그림 4. 초급 단계에서는 자주 쓰는 네 parameter 위치와 request body를 구분합니다. OAS 3.2에는 고급 `querystring` parameter 위치도 있습니다.

```
POST /orders/{orderId}?dryRun=true
Authorization: Bearer … · Cookie: locale=ko
```

PATH

orderId

resource 식별

QUERY

dryRun

조회·동작 선택

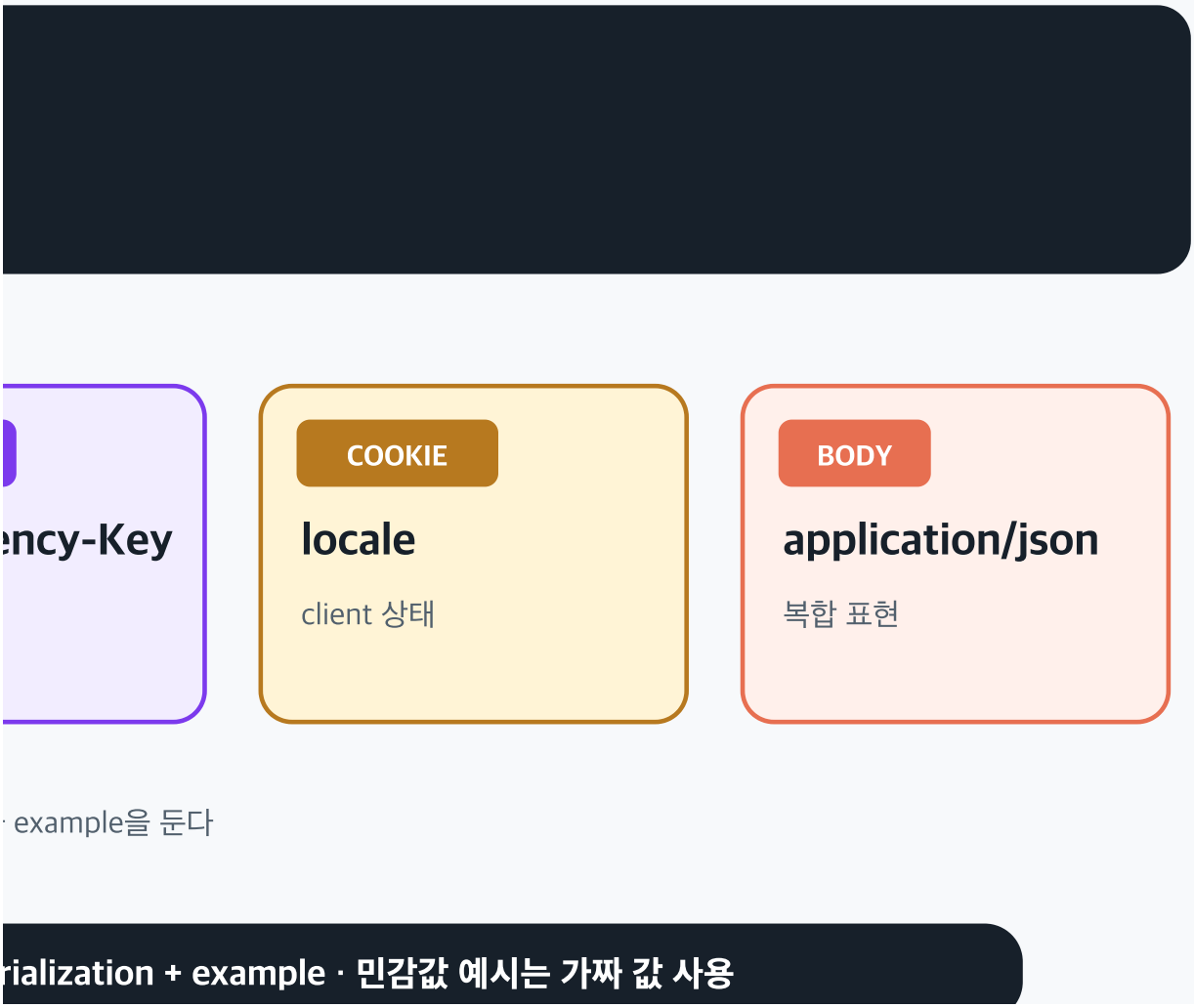
HEADER

Idempote

요청 문맥

path parameter는 template 이름과 일치하고 required: true · body는 media type별 schema와

입력 정의 = name + in + required + schema + se


ency-Key
COOKIE
locale
client 상태
BODY
application/json
복합 표현
· example을 둔다
rialization + example · 민감값 예시는 가짜 값 사용

OAS 3.2.0의 Parameter Object는 `path`, `query`, `querystring`, `header`, `cookie` 다섯 위치를 정의합니다. request body는 Parameter Object가 아니라 별도의 Request Body Object입니다.

5.1. path parameter

```
{
  "name": "shopId",
  "in": "path",
  "required": true,
  "schema": {"type": "string", "minLength": 1},
  "example": "shop_81"
}
```

path template의 `{shopId}`와 parameter `name`이 정확히 일치해야 합니다. `in: path`는 `required`가 반드시 `true`입니다.

5.2. query parameter

필터·정렬·페이지·표현 선택처럼 target resource를 읽는 조건에 자주 씁니다.

```
GET /orders?state=created&limit=20&cursor=abc
```

빈 값, 반복 값, array·object 직렬화는 `style`·`explode` 의미를 확인합니다. 이름과 타입만 적고 실제 URL 표현을 생략하면 client마다 다르게 보낼 수 있습니다.

5.3. header·cookie parameter

요청 문맥·조건·custom metadata를 표현할 수 있습니다. OAS Parameter Object에서 `Authorization`, `Content-Type`, `Accept`를 일반 header parameter처럼 정의하면 무시되는 규칙이 있으므로 security와 media type 구조를 사용합니다.

5.4. 고급 querystring

OAS 3.2.0의 `querystring`은 전체 URL query string을 하나의 값으로 content를 사용해 기술합니다. 같은 operation에서 `in: query`와 함께 쓸 수 없습니다. 초급 실습에서는 개별 `query` parameter를 사용하고, 복잡한 form query가 필요할 때 공식 serialization 규칙을 확인합니다.

6. request body는 content·schema·example의 세 겹입니다

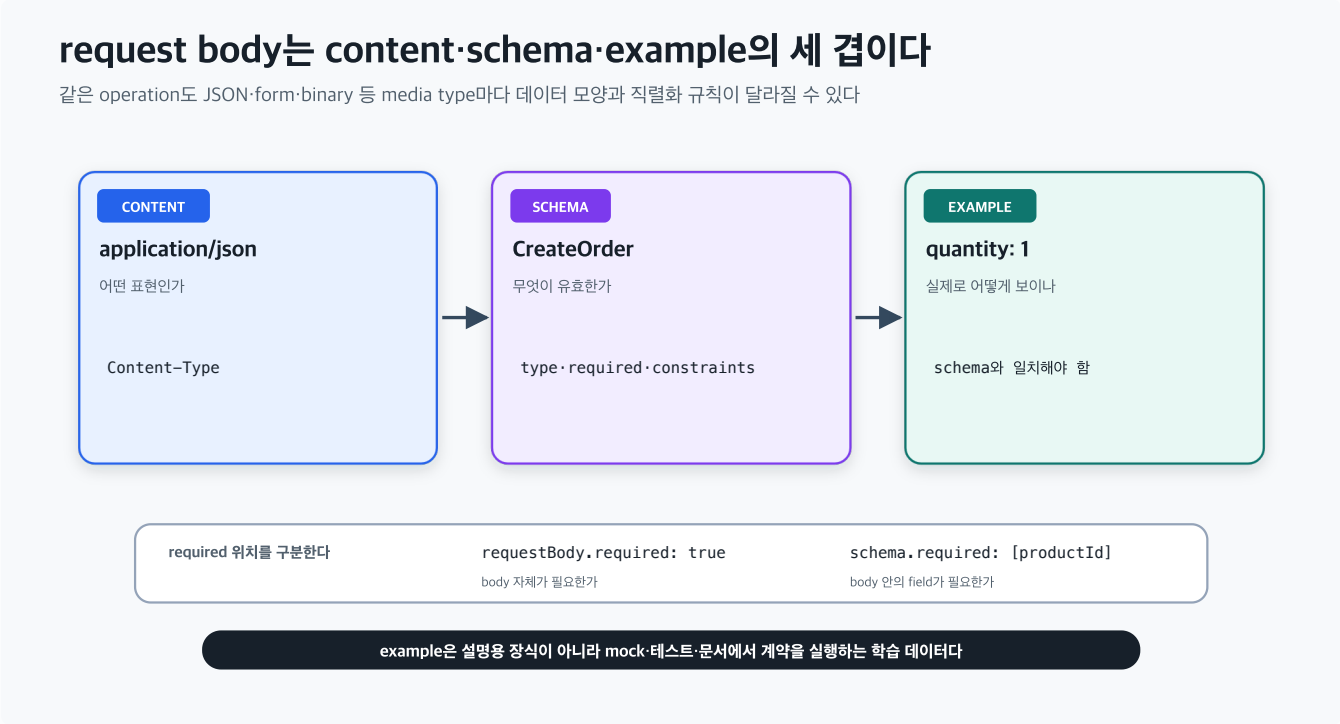
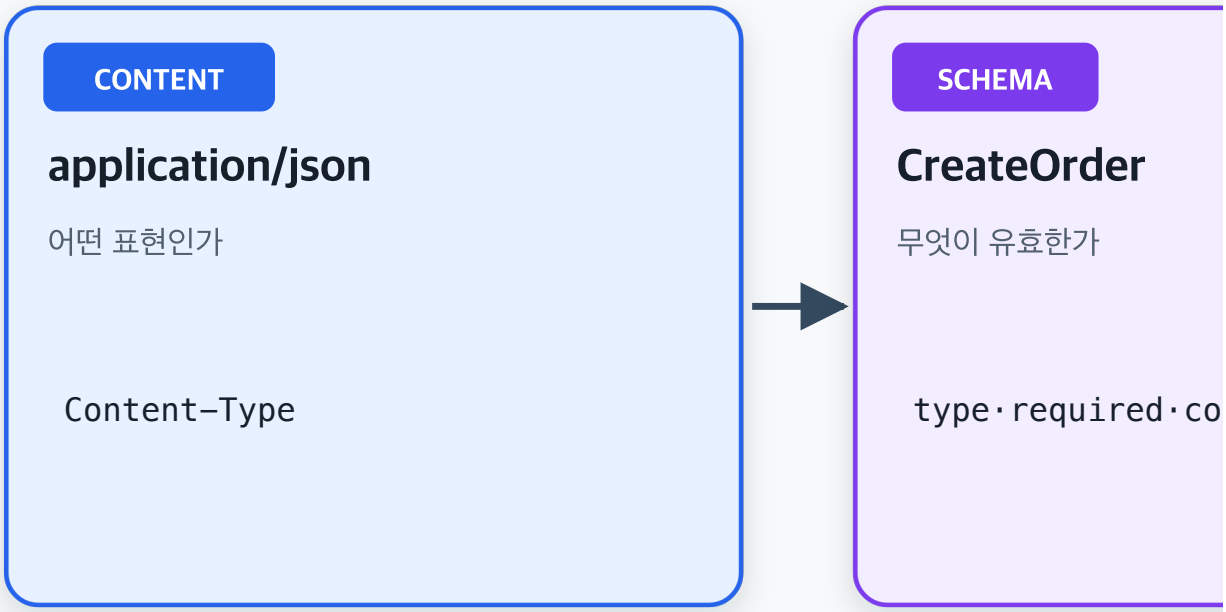


그림 5. media type이 표현 형식을, schema가 유효 범위를, example이 실제 instance를 보여 줍니다.

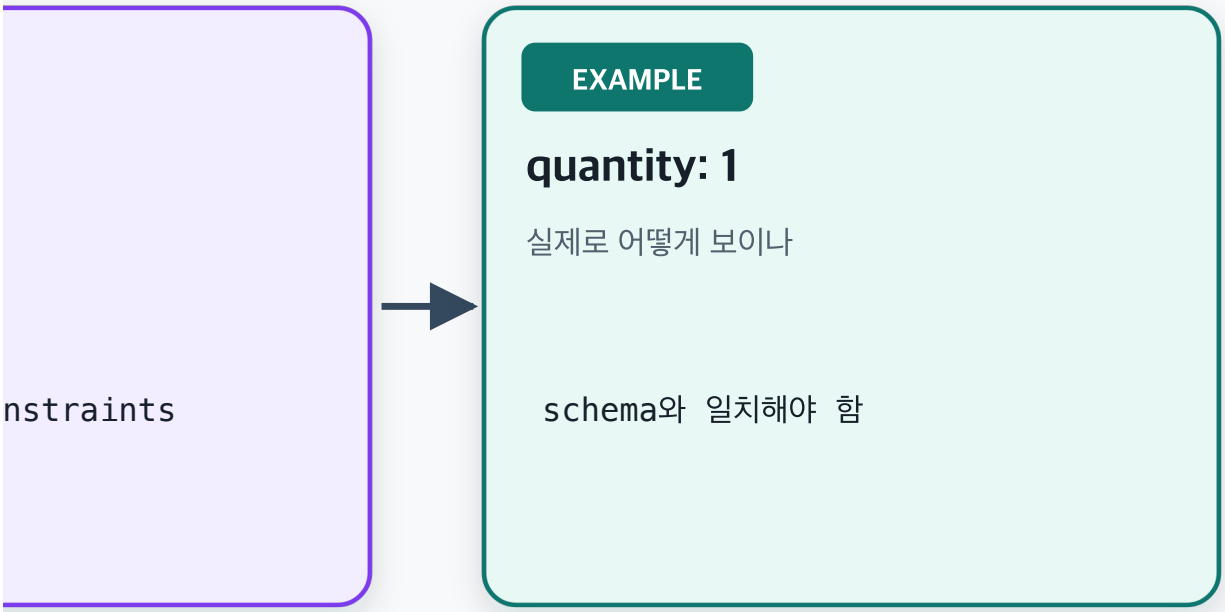


required 위치를 구분한다

`requestBody.required`

body 자체가 필요한가

example은 설명용 장식이 아니라 mock·테스



: true

schema.required: [productId]

body 안의 field가 필요한가

스트·문서에서 계약을 실행하는 학습 데이터다

```
"requestBody": {
  "required": true,
  "content": {
    "application/json": {
      "schema": {"$ref": "#/components/schemas/CreateOrder"},
      "example": {
        "productId": "course_ai_01",
        "quantity": 1
      }
    }
  }
}
```

6.1. 두 required는 위치가 다릅니다

위치	질문
`requestBody.required: true`	요청에 body 자체가 반드시 있어야 하는가
schema의 `required: [...]`	object 안에 어떤 property가 반드시 있어야 하는가

6.2. content key는 media type입니다

`application/json`, `multipart/form-data`, `text/csv`, `image/png`처럼 표현별 schema·encoding이 달라질 수 있습니다. JSON body를 받는다고 써 놓고 실제 example은 form field라면 계약이 모순됩니다.

6.3. example과 examples

Parameter·Media Type·Header Object에서 `example`과 `examples`는 동시에 쓸 수 없습니다. 여러 성공·오류·경계 사례가 필요하면 이름 있는 `examples`를 사용합니다. example은 schema와 media type·encoding에 맞아야 합니다.

7. JSON Schema로 데이터 경계를 씁니다

그림 6. property 목록만 쓰지 말고 type·필수·허용값·수치·문자열 경계를 함께 정의합니다.

```
type                          object
required                      [productId, quantity]
productId.type                string
productId.minLength           1
quantity.type                 integer
quantity.minimum              1
quantity.maximum              10
delivery.enum                 [email, download]
```

type

값의 JSON 종류

properties

객체의 알려진 필드

required

반드시 존재할 필드

enum

허용 값 집합

constraints

minimum·length·pattern

format

annotation·지원 확인

properties에 있다고 required는 아니다

format 검증 지원도 도구마다 확인한다

```
{
  "type": "object",
  "additionalProperties": false,
  "required": ["productId", "quantity"],
  "properties": {
    "productId": {"type": "string", "minLength": 1},
    "quantity": {"type": "integer", "minimum": 1, "maximum": 10},
    "delivery": {"type": "string", "enum": ["email", "download"]}
  }
}
```

7.1. properties와 required는 별개입니다

`properties`에 `delivery`가 있어도 `required` 배열에 없으면 생략 가능한 property입니다. 초보 명세에서 가장 흔한 오류 중 하나입니다.

7.2. additionalProperties 정책

알려지지 않은 field를 허용할지 명시적으로 검토합니다. 무조건 `false`가 정답은 아니지만, 요청 객체의 mass assignment·오타·호환성 정책과 연결됩니다.

7.3. format은 지원을 확인합니다

JSON Schema Draft 2020-12에서 `format`은 vocabulary 설정과 구현 지원에 따라 annotation 또는 assertion으로 다뤄질 수 있습니다. `format: email`만 쓰고 runtime이 반드시 거절한다고 가정하지 않습니다. 실제 validator 설정과 contract test를 확인합니다.

7.4. OpenAPI dialect

OAS 3.2는 Schema Object에서 JSON Schema dialect를 사용하며 root의 `jsonSchemaDialect`로 기본 dialect를 정할 수 있습니다. 문서가 사용하는 OAS version·dialect와 도구 지원 범위를 함께 고정합니다.

30초 확인

`quantity: 0` example이 있는데 schema는 `minimum: 1`이라면 둘 중 하나가 거짓입니다. 문서 화면이 예쁘게 렌더링돼도 계약은 실패입니다.

8. responses는 성공과 알려진 오류를 함께 닫습니다

그림 7. status마다 body 모양뿐 아니라 완료 의미와 소비자가 취할 다음 행동까지 연결합니다.

STATUS	MEANING	HEADER·BODY
201	Created	Location + Orde
400	Invalid field	Problem + error
401	Authentication	Problem
403	Forbidden	Problem
409	Conflict	Problem + curre
503	Unavailable	Problem + retry

	완료·다음 행동
`r`	주문 resource 생성
`s[]`	입력 수정
	재로그인
	권한 확인
`nt`	상태·수량 변경
	안전한 재시도

OAS 3.2 Responses Object는 operation의 예상 response를 HTTP status code와 매핑합니다. 적어도 하나의 response가 있어야 하며 성공 response와 알려진 오류를 문서화할 것이 기대됩니다.

```
"responses": {
  "201": {"description": "주문 생성 완료"},
  "400": {"$ref": "#/components/responses/InvalidField"},
  "401": {"$ref": "#/components/responses/AuthenticationRequired"},
  "403": {"$ref": "#/components/responses/Forbidden"},
  "409": {"$ref": "#/components/responses/StockConflict"},
  "503": {"$ref": "#/components/responses/Unavailable"}
}
```

YAML에서도 status code key를 문자열로 해석하도록 따옴표를 쓰는 것이 OAS 규칙과 호환됩니다.

8.1. 200·201·202·204를 구분합니다

status	대표 의미	명세에서 볼 것
200	성공 결과 representation	body schema·완료 범위
201	새 resource 생성	Location·resource ID·representation
202	접수했지만 처리 미완료	job ID·상태 확인·최종 완료
204	성공, response content 없음	화면 완료 증거·header

RFC 9110에서 201은 요청이 수행되어 하나 이상의 새 resource가 생성됐음을 뜻합니다. 202는 처리를 접수했지만 아직 끝나지 않았고 나중에 실행되지 않을 수도 있습니다.

8.2. response는 description만이 아닙니다

field	역할
`description`	response 의미 설명
`headers`	Location·Retry-After·pagination 등 header 계약
`content`	media type별 schema·example
`links`	response 값에서 다른 operation으로 이어지는 설계 시점 관계

8.3. default와 range

`default`는 개별 정의하지 않은 status의 기본 response를 표현할 수 있습니다. OAS 3.2는 `2XX`, `4XX`, `5XX` range도 허용하지만, 구체적인 status가 있으면 그것이 우선합니다. 중요한 known error를 range 하나로 숨기지 않습니다.

9. 오류는 problem details family로 일관되게 씁니다

그림 8. 오류마다 임의 문자열을 만들지 않고 공통 뼈대와 안정적인 problem type을 공유합니다.

```
type          /problems/stock-conflict

title         재고가 부족합니다

status        409

detail        수량을 3 이하로 바꾸세요

instance      /problems/prb-81

errors        [{field: quantity, max: 3}]
```

invalid-field

400 · errors[]

authentication

401 · login

forbidden

403 · policy

stock-conflict

409 · current

unavailable

503 · retry

detail 문자열 대신 type·확장 field로 기계 분기

```
{
  "type": "https://yeoncore.ai/problems/stock-conflict",
  "title": "재고가 부족합니다",
  "status": 409,
  "detail": "수량을 3 이하로 바꾸세요.",
  "instance": "/problems/prb-81",
  "errors": [{"field": "quantity", "max": 3}]
}
```

RFC 9457은 `application/problem+json`과 `type`, `title`, `status`, `detail`, `instance`를 정의합니다. `detail`은 사람이 이번 오류를 고치게 돕고, client code는 문장 parsing 대신 안정적인 `type`과 구조화 확장 field를 사용합니다.

9.1. known error 표

조건	status·type	소비자 행동
field 누락·범위	400 invalid-field	해당 field 수정
인증 없음·만료	401 authentication-required	재인증
객체·기능 권한 없음	403 forbidden	권한 확인·문의
현재 상태 충돌	409 stock-conflict	수량·상태 변경
의존 서비스 일시 실패	503 unavailable	retry 조건 확인

9.2. 같은 schema, 다른 example

`Problem` schema를 재사용하더라도 각 status·type별 example은 실제 소비자가 보게 될 field·다음 행동을 보여 줍니다. 모든 오류를 `message: string` 하나로 줄이면 기계 분기·field 오류·문의 추적을 잃습니다.

10. security는 scheme·requirement·scope를 연결합니다

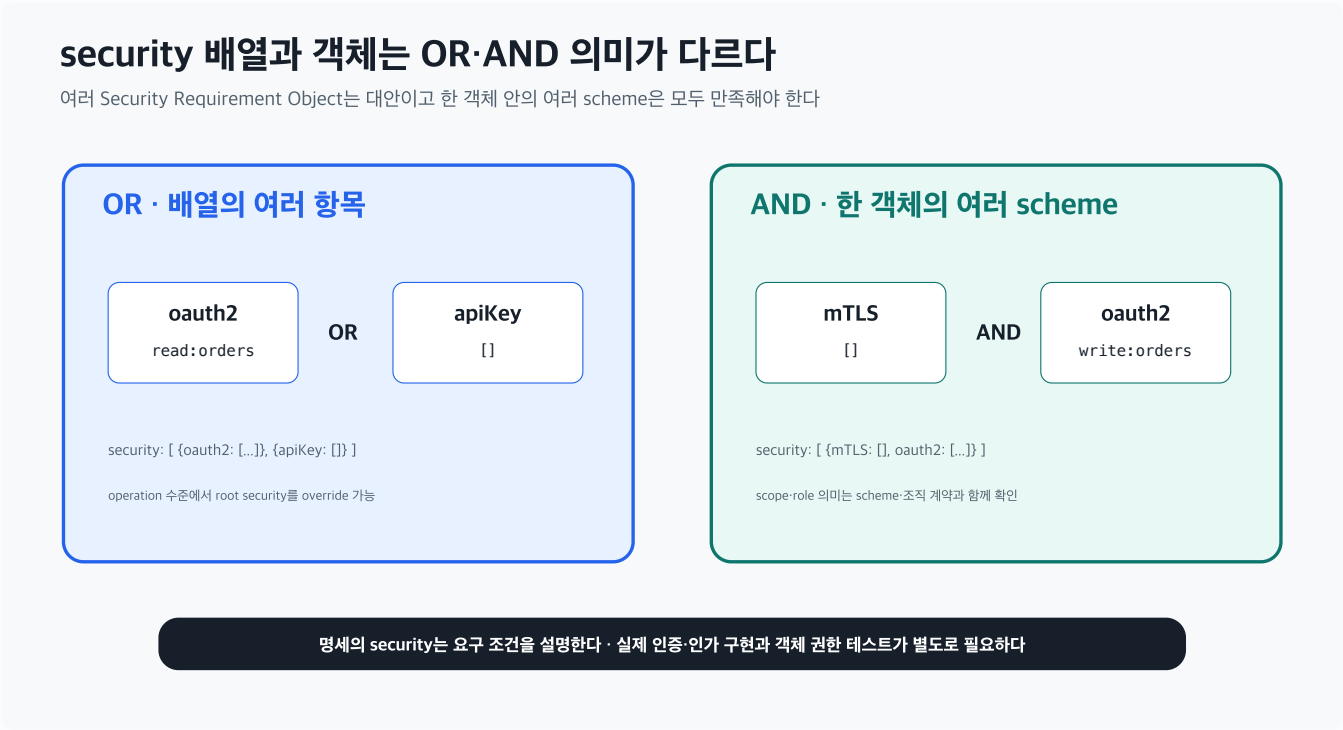


그림 9. 배열의 여러 Security Requirement Object는 대안 OR이고, 한 객체 안의 여러 scheme은 함께 만족해야 하는 AND입니다.

OR · 배열의 여러 항목

oauth2

`read:orders`

OR

apiKey

`[]`

security: [{oauth2: [...]}, {apiKey: []}]

operation 수준에서 root security를 override 가능

명세의 security는 요구 조건을 설명한다 · 실제 인

AND · 한 객체의 여러 scheme

mTLS

[]

AND

oauth2

`write:orders`

security: [{mTLS: [], oauth2: [...]}]

scope·role 의미는 scheme·조직 계약과 함께 확인

증·인가 구현과 객체 권한 테스트가 별도로 필요하다

10.1. security scheme 정의

```
"securitySchemes": {
  "oauth2": {
    "type": "oauth2",
    "flows": {
      "authorizationCode": {
        "authorizationUrl": "https://auth.example/authorize",
        "tokenUrl": "https://auth.example/token",
        "scopes": {"write:orders": "주문 생성"}
      }
    }
  }
}
```

10.2. operation의 requirement

```
"security": [
  {"oauth2": ["write:orders"]}
]
```

root security는 기본 요구이고 operation security는 이를 override할 수 있습니다. 빈 requirement `{}`를 배열에 넣으면 security optional 의미를 만들 수 있으므로 의도치 않은 공개 여부를 검수합니다.

10.3. 문서화와 구현은 별개입니다

security requirement를 적었다고 runtime 인증·인가가 생기지 않습니다. 401·403 response, 객체 수준 권한, scope·role 정책, 테스트를 함께 둡니다.

> **secret을 example에 넣지 않습니다.**
> 실제 bearer token·API key·cookie·개인정보를 명세·mock·스크린샷에 넣지 않습니다. 형식만 보여 주는 명백한 가짜 값을 사용합니다.

11. components와 $ref로 계약 조각을 재사용합니다

그림 10. 공통 schema·parameter·response·security scheme을 component로 두고 여러 operation에서 reference합니다.


components
schemas
Order·Problem
parameters
OrderId
responses
StockConflict
securitySchemes
oauth2
재사용은 복사 감소뿐 아니라 오류 형식과 식별

$ref

T /orders/{id}

omponents/schemas/Order

$ref

ST /orders

omponents/responses/StockConflict

$ref

LETE /orders/{id}

omponents/parameters/OrderId

자 규칙을 여러 operation에서 일관되게 만든다

```
"schema": {"$ref": "#/components/schemas/Order"}
```

11.1. 재사용하기 좋은 것

- 모든 오류가 공유하는 `Problem`
- 여러 operation의 `OrderId` path parameter
- 반복하는 `AuthenticationRequired` response
- OAuth2·API key 같은 security scheme
- 여러 response에서 쓰는 `Order` · `PageInfo`

11.2. 너무 이른 추상화는 읽기를 어렵게 합니다

한 번만 쓰는 작은 schema까지 깊은 `$ref` 사슬로 만들면 operation을 따라가기 어렵습니다. 반복·정합성·독립 변경 경계를 기준으로 component를 뽑습니다.

11.3. reference도 versioned contract입니다

공통 `Problem`의 required field를 바꾸면 이를 참조하는 모든 error response에 영향이 갑니다. component 변경도 소비자 영향 검토와 contract test가 필요합니다.

12. example은 계약을 실행하는 학습 데이터입니다

좋은 example은 happy path 한 개만 보여 주지 않습니다.

example 이름	보여 줄 것
minimal	필수 field만 있는 가장 작은 유효 요청
full	선택 field까지 포함한 유효 요청
boundary	minimum·maximum·빈 목록 등 경계값
invalid-field	field 오류와 수정 단서
conflict	현재 상태 충돌
async-accepted	job ID·status URL

12.1. example 검수

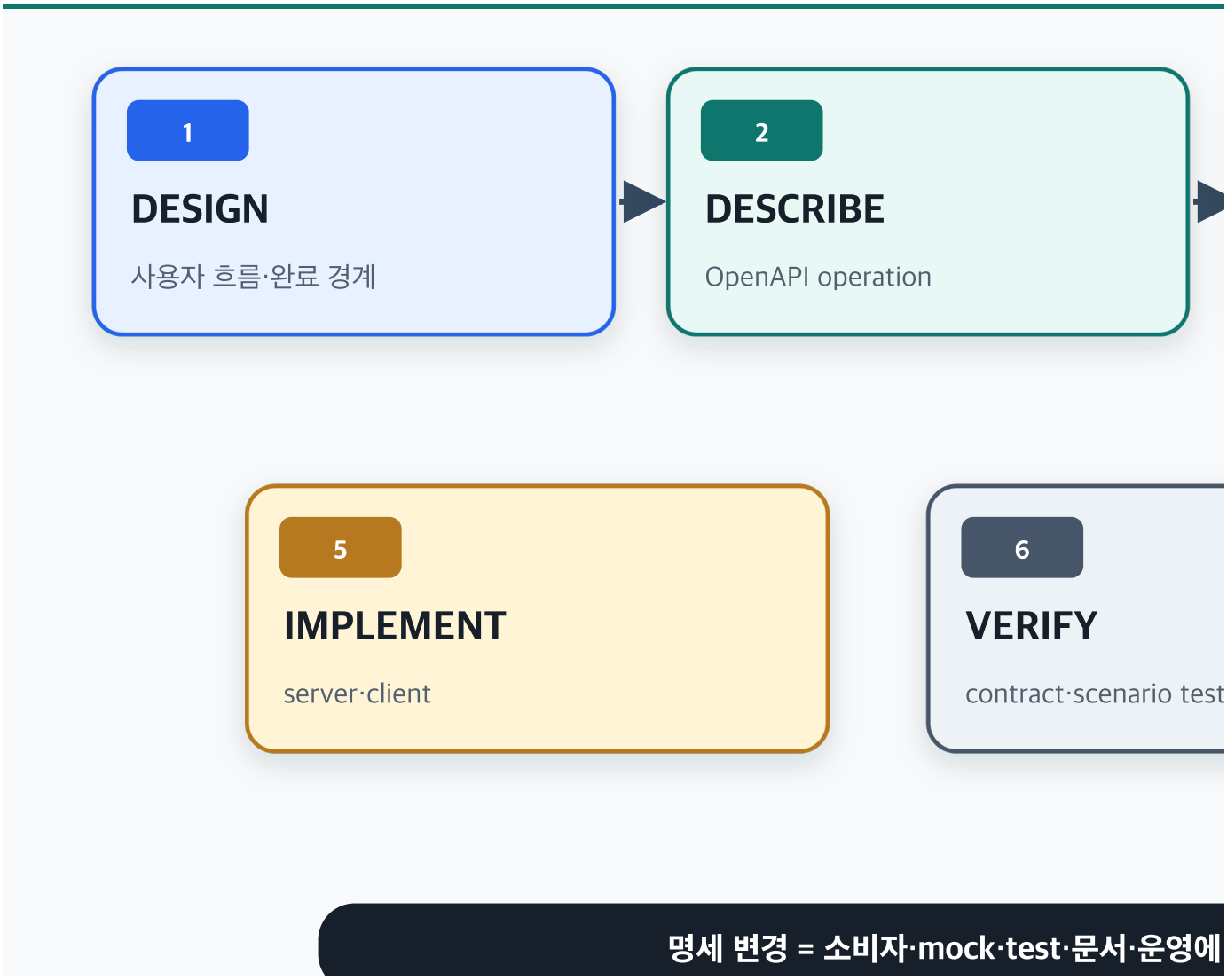
```
JSON parse 가능?
→ media type과 일치?
→ schema type·required·constraints 통과?
→ 실제로 존재하는 enum 값?
→ 실제 secret·개인정보 없음?
→ response의 완료 의미와 일치?
```

12.2. example과 mock의 차이

example은 계약 안의 instance입니다. mock server는 그 example·schema로 가짜 응답을 제공해 client 흐름을 먼저 시험할 수 있습니다. mock 성공은 실제 인증·업무 규칙·transaction 구현 성공을 뜻하지 않습니다.

13. 명세를 설계·구현·검수의 공통 입력으로 씁니다

그림 11. 사용자 완료 경계에서 시작해 명세·자동 검사·mock·구현·contract test·운영 증거까지 같은 계약을 이어 씁니다.


1
DESIGN
사용자 흐름·완료 경계
2
DESCRIBE
OpenAPI operation
5
IMPLEMENT
server·client
6
VERIFY
contract·scenario test
명세 변경 = 소비자·mock·test·문서·운영에

미치는 호환성 영향까지 검토하는 제품 변경

13.1. contract-first와 code-first

접근	시작	장점	주의
contract-first	합의한 명세	client·server 병렬 협업·초기 mock	구현과 drift 방지 필요
code-first	구현·annotation에서 생성	기존 코드 문서화 속도	사용자·소비자 관점이 늦을 수 있음

둘 중 하나가 항상 우월하지 않습니다. 어느 접근이든 최종 기준 문서와 변경 절차, runtime 검증을 정합니다.

13.2. 검사 층

1. JSON·YAML parse
2. 선언한 OAS version 구조 검사
3. style·naming·조직 규칙 lint
4. schema와 example validation
5. mock으로 client flow 검토
6. provider·consumer contract test
7. production status·problem·trace 관측

문서 렌더링 성공은 계약 검증 성공이 아닙니다.
도구가 예쁜 API 문서를 만들었어도 example이 schema를 어기거나 보호된 operation에 security가 없거나 구현이 다른 status를 반환할 수 있습니다.

14. 실습 · 여덟 명세를 자동 진단합니다

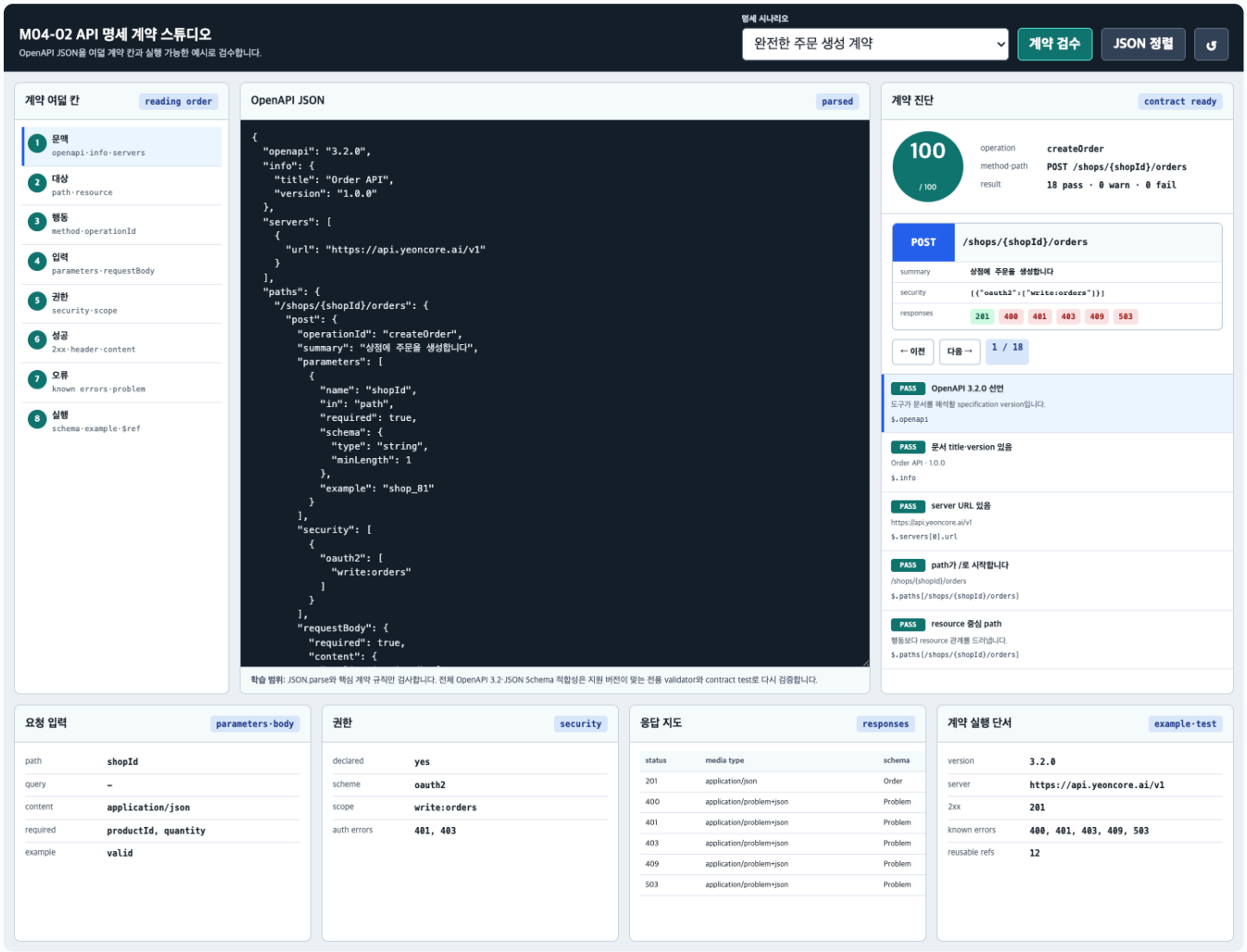


그림 12. 실습기는 OpenAPI JSON을 읽어 endpoint 카드·입력·security·responses·18개 핵심 검사로 동시에 보여 줍니다.

14.1. 준비 파일

- 실습 안내
- API 명세 계약 스튜디오
- API endpoint 명세 템플릿
- API 명세 용어집

14.2. 여덟 시나리오

시나리오	핵심 결함	고칠 위치
완전한 주문 생성	없음	18 pass
path parameter 누락	`{shopId}` 계약 없음	operation parameters
schema·example 불일치	empty·minimum·enum 위반	request example 또는 schema
성공 response만 있음	known error·auth error 없음	responses
권한 미기재	security·401·403 없음	security·responses
문자열 오류	problem details 없음	error response content
202 추적 누락	job ID·status URL 없음	202 schema·example
operationId 중복	두 operation 같은 ID	operationId naming

14.3. 직접 수정할 것

1. path parameter를 지운 시나리오에 `shopId` 정의를 복구합니다.
2. example의 `quantity`를 1로, `delivery`를 enum 값으로 바꿉니다.
3. 202 response에 `jobId`와 `statusUrl` properties·example을 추가합니다.
4. 중복 operationId를 `getOrder`로 바꿉니다.
5. 수정할 때마다 `계약 검수`를 다시 누릅니다.

실습 통과 기준
JSON 문법 오류, path parameter 누락, example mismatch, security 누락, known error 부족, problem 형식 불일치, 202 추적 누락, operationId 중복 중 여섯 가지 이상을 위치·이유·수정으로 설명합니다.

15. 실무 예제 · 교육 신청 생성

15.1. endpoint 카드

칸	계약
문맥	OpenAPI 3.2.0·Education API 1.0.0·sandbox server

칸	계약
대상	`/courses/{courseId}/applications`
행동	POST·`createApplication`
입력	courseId path·application/json body
권한	oauth2 `write:applications`
성공	201·Location·Application
오류	400·401·403·409·503 problem
예시·재사용	minimal·duplicate·components refs

15.2. 요청·응답

```
{
  "applicantName": "김연코어",
  "email": "learner@example.com",
  "motivation": "업무 자동화 서비스를 기획하고 싶습니다."
}
```

```
{
  "id": "app_81",
  "courseId": "course_ai_01",
  "state": "submitted",
  "submittedAt": "2026-07-15T19:00:00+09:00"
}
```

15.3. 중복 신청

```
{
  "type": "https://yeoncore.ai/problems/duplicate-application",
  "title": "이미 신청했습니다",
  "status": 409,
  "detail": "기존 신청 app_72를 확인하세요.",
  "instance": "/problems/prb-72",
  "existingApplicationId": "app_72"
}
```

16. 변경 호환성을 별도 검수합니다

명세는 현재 모양뿐 아니라 시간에 따른 계약입니다.

변경	위험	먼저 할 일
required request field 추가	기존 client 요청 실패	optional·default·새 version 검토
enum 값 삭제	기존 저장값·client 분기 실패	사용량·전환 기간 확인
response field 삭제·rename	client parsing 실패	추가 후 deprecation·migration
status 의미 변경	화면 분기·retry 오류	새 problem type·명확한 변경 공지
security scope 강화	기존 client 403	권한 전환·발급 절차
schema 제약 강화	이전 유효값 거절	production 데이터·consumer test

`info.version` 숫자만 올리고 끝내지 않습니다. 누가 소비하고, 어떤 example·mock·SDK·test·운영 규칙이 영향을 받는지 기록합니다.

그림 13. 한 장 요약. server에서 시작해 example·test까지 한 방향으로 읽으면 endpoint의 모호한 빈칸이 보입니다.

1

문맥

openapi·server

2

대상

path·resource

5

권한

security·scope

6

성공

status·header·body

읽기 순서 · server → path → method → inputs

완료 기준 = 성공·known error·권한·예시

3

행동

method·operationId

4

입력

parameters·body

7

오류

problem family

8

실행

schema·example·test

→ security → responses → examples → refs

가 같은 schema와 업무 완료 의미를 말한다

17. 인쇄용 endpoint 워크시트

17.1. 문맥·대상·행동

```
OAS version: _____________________________________________
API title·info.version: ___________________________________
server·environment: _______________________________________
resource·path template: ___________________________________
HTTP method·operationId: __________________________________
사용자 결과·summary: _____________________________________
```

17.2. 입력·권한

위치	name	required	schema·serialization	example
path		true		
query				
header				
cookie				

```
request body required: ____________________________________
media type: _______________________________________________
schema ref·required fields: _______________________________
security scheme·scope: ____________________________________
```

17.3. response

status	의미	headers	media·schema	example	소비자 다음 행동
2xx					
400					
401					

status	의미	headers	media·schema	example	소비자 다음 행동
403					
409					
5xx					

18. 셀프 테스트 · 정답을 가리고 풉니다

문제 1

`openapi: 3.2.0`과 `info.version: 1.4.0`은 각각 무엇을 뜻합니까?

문제 2

전체 호출 URL `https://api.example/v1/orders/81?include=receipt`에서 server, path, path parameter, query를 나눕니다.

문제 3

`/orders/{orderId}`에 반드시 필요한 Parameter Object의 세 field를 씁니다.

문제 4

OAS 3.2.0의 parameter 위치 다섯 가지와 request body의 차이를 씁니다.

문제 5

`requestBody.required`와 Schema Object의 `required`는 무엇이 다릅니까?

문제 6

Schema Object의 `properties`에 field가 있으면 그 field는 자동으로 필수입니까?

문제 7

`format: email`을 썼으니 모든 validator가 잘못된 email을 거절한다고 가정해도 됩니까?

문제 8

201과 202가 각각 보장하는 완료 의미와 명세에 넣을 추적 단서를 씁니다.

문제 9

보호된 endpoint에 성공 200만 있고 401·403이 없습니다. 무엇이 빠졌습니까?

문제 10

Security Requirement 배열에 `{oauth2: [...]}`와 `{apiKey: []}`가 별도 항목으로 있으면 OR입니까 AND입니까?

문제 11

`example`의 `quantity`가 0인데 schema minimum은 1입니다. 무엇을 고쳐야 합니까?

문제 12

두 operation이 같은 `operationId`를 가지면 어떤 문제가 생깁니까?

19. 셀프 테스트 정답과 해설

정답 1

`openapi`는 문서를 해석할 OAS version이고 `info.version`은 API description·서비스 계약의 version입니다. 서로 자동으로 같지 않습니다.

정답 2

server는 `https://api.example/v1`, path template은 `/orders/{orderId}`, path parameter 값은 `81`, query는 `include=receipt`입니다.

정답 3

`name: orderId`, `in: path`, `required: true`입니다. 여기에 type을 가진 `schema`도 필요합니다.

정답 4

`path`, `query`, `querystring`, `header`, `cookie` 입니다. request body는 Parameter Object가 아니라 media type별 content를 가진 Request Body Object입니다.

정답 5

앞은 body 자체가 필요한지, 뒤는 object 안에서 반드시 존재할 property를 정합니다.

정답 6

아닙니다. `required` 배열에 포함되어야 필수입니다.

정답 7

안 됩니다. format vocabulary와 validator 지원·설정을 확인하고 실제 validation test를 해야 합니다.

정답 8

201은 새 resource 생성을 뜻하므로 Location·resource ID·representation을 검토합니다. 202는 접수했지만 처리 미완료이므로 job ID·status URL·최종 완료 확인 방법이 필요합니다.

정답 9

인증 필요와 권한 거절의 response 계약, problem type·schema·example·소비자 행동이 빠졌습니다. security requirement도 함께 확인합니다.

정답 10

OR입니다. 둘 중 한 Security Requirement Object를 만족하면 됩니다. 한 객체 안에 두 scheme이 있으면 AND입니다.

정답 11

업무 의도에 맞는 쪽을 고칩니다. 유효 요청 example이라면 quantity를 1 이상으로 바꿉니다. 0을 허용해야 한다면 schema·업무 규칙을 함께 다시 합의합니다.

정답 12

code generation·link·test·문서 도구가 operation을 고유하게 식별하지 못해 충돌하거나 잘못 연결될 수 있습니다.

20. 최종 제출 체크리스트와 공식 근거

- [ ] 선언한 OAS version을 사용 도구가 지원합니다.
- [ ] `openapi`와 `info.version`의 역할을 구분했습니다.
- [ ] server·path·method·operationId가 일관됩니다.
- [ ] 모든 path placeholder에 `in:path, required:true` parameter가 있습니다.
- [ ] query·header·cookie의 serialization을 검토했습니다.
- [ ] request body에 media type·schema·유효한 example이 있습니다.
- [ ] properties와 required를 구분했습니다.
- [ ] 2xx response가 업무 완료 의미를 말합니다.
- [ ] 201에 생성 resource, 202에 상태 추적 단서가 있습니다.
- [ ] 400·401·403·409·5xx known error를 필요에 맞게 정의했습니다.
- [ ] 오류가 안정적인 problem type과 구조화 field를 사용합니다.
- [ ] security scheme·requirement·scope와 401·403을 연결했습니다.
- [ ] operationId가 전체 API에서 고유합니다.
- [ ] example에 secret·개인정보가 없습니다.
- [ ] component 변경의 소비자 영향을 검토했습니다.
- [ ] 전용 validator·mock·contract test로 runtime과 비교했습니다.

이 교재는 다음 공식 자료를 2026-07-15에 확인해 학습자용으로 재구성했습니다.

- OpenAPI Specification 3.2.0: OpenAPI Object·Paths·Operation·Parameter·Request Body·Responses·Security·Examples·Components
- OpenAPI Specification versions: version 목록과 schema iterations
- JSON Schema Draft 2020-12: core·validation vocabulary
- RFC 9110 · HTTP Semantics: method·status·201·202 의미
- RFC 9457 · Problem Details for HTTP APIs: application/problem+json과 오류 멤버

M04-02 완료
이제 API 명세를 URL 목록이 아니라 실행 가능한 계약으로 읽을 수 있습니다. endpoint 하나를 골라 여덟 칸을 채우고, 유효 request example과 success·known error example을 schema에 통과시키면 기획·개발·테스트가 같은 결과를 말할 수 있습니다.