v 3.0

백엔드 API 설계 지침서

지속 가능한 서버 인터페이스를 만들기 위한
원칙, 패턴, 체크리스트, 예시

ENDPOINT DESIGN REFERENCE

GET	/users/{id}	200	401
POST	/orders	201	404
PUT	/users/{id}	200	422
PATCH	/users/{id}	200	400
DELETE	/orders/{id}	204	403

COVERS 21 CHAPTERS

자원 중심 설계 · 인증 & 보안 · 에러 모델 · 버전 관리 · 멱등성 · 관측성 · 테스트 전략

error_response.json

```
{
  "error": {
    "code":      "INVALID_PARAMETER",
    "message":   "field validation failed",
    "requestId": "req_01HQX7K...",
  }
}
```

목 차

Table of Contents

"좋은 API 는 특별한 천재성보다 기본을 지키는 힘에서 나온다."

들어가며

API 는 단순히 클라이언트와 서버가 데이터를 주고받는 통로가 아니다. API 는 서비스의 계약(contract)이며, 조직의 개발 문화와 운영 역량을 반영하는 경계면이다. 잘 설계된 API 는 프론트엔드, 모바일, 외부 제휴사, 내부 운영도구, 데이터 파이프라인이 안정적으로 협업하도록 돕는다.

반대로 즉흥적으로 설계된 API 는 같은 기능을 두고도 중복 구현, 예외 처리 남발, 인증 혼선, 문서 부재, 버전 충돌, 운영 사고를 유발한다.

◆ 이 지침서의 목적

이 지침서는 백엔드 API 를 설계할 때 필요한 핵심 기준을 한 권으로 정리하기 위해 작성되었다. 특정 프레임워크에 종속되지 않으며, REST 스타일 API 를 중심에 두되 실제 현업에서 자주 마주치는 인증, 상태 코드, 에러 모델, 페이징, 멱등성, 비동기 처리, 보안, 문서화, 변경 관리, 관측성까지 함께 다룬다.

◆ 대상 독자

- 처음으로 백엔드 API 를 설계하는 개발자
- 여러 팀이 함께 쓰는 공통 API 규칙을 정리하려는 리드 개발자
- 외부 공개 API 또는 파트너 API 를 설계하는 아키텍트
- 운영 안정성과 변경 관리까지 포함한 설계 기준이 필요한 조직

> *이 문서의 목표는 "멋져 보이는 설계"가 아니라 "오래 버티는 설계"를 만드는 것이다.*

CH 01

API 설계의 목적과 철학

좋은API 란 무엇인가

1.1 API 는 기능이 아니라 계약이다

API 를 설계한다는 것은 URL 몇 개를 정하는 작업이 아니다. API 는 어떤 자원을 다루는가, 어떤 입력을 받는가, 어떤 출력을 돌려주는가, 실패했을 때 어떻게 설명하는가 — 이 네 가지를 동시에 정의하는 계약이다.

계약으로서의 API 는 클라이언트가 서버 내부 구현을 몰라도 안정적으로 사용할 수 있어야 한다. 따라서 API 설계에서는 내부 코드 구조보다 외부 사용 경험이 더 중요하다.

> 💡 **설계 원칙**
> 데이터베이스 테이블 구조를 그대로 노출하는 API 는 구현은 쉽지만 계약으로는 약하다. 도메인 행위와 사용 시나리오를 중심으로 설계한 API 는 내부 구현이 바뀌어도 외부 계약을 오래 유지할 수 있다.

1.2 좋은 API 의 기준

기준	설명
예측 가능하다	비슷한 자원은 비슷한 규칙으로 동작한다
일관적이다	이름, 상태 코드, 에러 형식, 페이징 방식이 통일되어 있다
명확하다	필수값, 선택값, 제약조건이 문서와 응답에 분명히 드러난다
확장 가능하다	미래의 필드 추가, 정책 변경, 기능 확장에 견딜 수 있다
안전하다	인증, 권한, 입력 검증, 민감정보 보호가 고려되어 있다
운영 가능하다	로그, 추적, 모니터링, 재시도, 장애 대응이 가능하다

1.3 API 설계에서 흔한 실패 원인

- URL 이 기능명 중심으로 난립한다. 예: /getUserInfo, /updateUserDataNow
- 성공/실패 형식이 API 마다 다르다
- HTTP 상태 코드를 무시하고 항상 200 만 반환한다
- 에러 메시지가 사람이 읽을 수 없거나, 반대로 내부 정보를 과도하게 노출한다
- 같은 의미의 필드가 API 마다 이름이 다르다. 예: userId, uid, member_id
- 문서보다 구현이 먼저 바뀌고, 문서는 방치된다
- 한 번 공개한 필드를 제거하거나 의미를 바꿔 클라이언트를 깨뜨린다

1.4 설계 우선순위

순위	기준	설명
1	도메인 의미가 맞는가	자원과 행위가 도메인 언어로 정확히 표현되는가

순위	기준	설명
2	외부 계약이 일관적인가	소비자 입장에서 예측 가능하고 일관된 구조인가
3	오류 상황까지 설명 가능한가	실패 케이스가 구조적으로 설명되는가
4	확장과 운영을 견딜 수 있는가	변경과 운영 요구에 견고한 구조인가
5	구현이 편한가	구현 편의성은 중요하지만 최우선이 되어서는 안 된다

CH 02

설계 시작 전 반드시 정해야 할 것들

API 설계의 사전 조건

2.1 API 의 사용자 정의

API 를 설계하기 전에 먼저 이 API 의 소비자가 누구인지 정의해야 한다.

소비자 유형	설계 시 중점
웹 프론트엔드	JSON 구조, CORS 정책, 인증 토큰 처리
모바일 앱	응답 크기 최적화, 오프라인 대응, 버전 호환
외부 파트너	하위 호환성, 문서화, 버전 정책, SLA
사내 백오피스	내부 권한 모델, 감사 로그, 배치 처리
마이크로서비스	서비스 간 계약, 멱등성, 서킷 브레이커

⚠ **주의**
내부 API 라 해도 팀이 여러 곳이면 사실상 공개 API 처럼 관리해야 한다.

2.2 도메인 경계 정의

API 는 데이터베이스 엔티티가 아니라 도메인 자원 기준으로 설계해야 한다. 쇼핑몰이라면 사용자, 상품, 장바구니, 주문, 결제, 배송 같은 자원이 중심이 된다.

💡 **테이블 직노출 금지**
order_items 테이블이 있다고 해서 외부에 /order-items 가 반드시 필요한 것은 아니다. 외부 계약은 "주문" 개념 안에서 더 자연스럽게 표현될 수 있다.

2.3 표준 규칙 문서 선행

개별 API 를 만들기 전에 최소한 다음 규칙은 팀 내 표준으로 먼저 정해 두는 것이 좋다: URL 네이밍, JSON 필드 네이밍, 날짜/시간 포맷, 상태 코드 기준, 공통 에러 포맷, 인증 방식, 페이지네이션 방식, 버전 정책, 문서화 기준.

2.4 변경 가능성과 안정성의 균형

- 기존 필드 의미를 바꾸지 않는다
- 필드 삭제보다 deprecated 처리 후 유예를 택한다
- 선택 필드 추가는 비교적 안전하지만, 필수 필드 추가는 신중해야 한다
- 응답 구조의 대변경은 버전 분리를 검토한다

CH 03

자원 중심 설계

Resource-Oriented Design

3.1 자원(Resource) 사고방식

REST 스타일 설계의 핵심은 자원 중심 사고다. URL 은 동사를 나열하는 대신 "무엇"을 다루는지 표현해야 한다.

```
# ✅ 좋은 예 — 자원 중심
GET  /users/{userId}
GET  /orders/{orderId}
POST /orders
POST /orders/{orderId}/cancel

# ❌ 나쁜 예 — 동사 중심
POST /getUser
POST /createNewOrderNow
POST /orderCancelProcess
```

3.2 컬렉션과 단건 자원

- 컬렉션: 복수형 명사 — /users
- 단건: 식별자 포함 — /users/{userId}
- 하위 자원은 소유 관계가 분명할 때만 사용 — /orders/{orderId}/items
- 하위 자원이 독립적 수명주기를 가지면 최상위 자원으로 분리

3.3 행위(Action)의 표현

모든 행위를 CRUD 로만 표현할 수는 없다. 주문 취소, 이메일 인증 발송 같은 행위는 액션 엔드포인트가 더 자연스럽다.

```
POST /orders/{orderId}/cancel
POST /users/{userId}/email-verification
POST /auth/password-reset-requests
```

💡 **선택 기준**
상태 전이가 핵심이면 액션 엔드포인트를, 단순 속성 변경이면 PATCH/PUT 을 우선 고려한다.

3.4 URL 설계 원칙

원칙	권장	지양
소문자 사용	/payment-methods	/Payment-Methods
하이픈 구분	/shipping-addresses	/shipping_addresses
확장자 제거	/orders	/orders.json
명사 사용	/access-tokens	/doLogin
깊이 제한 (3 단계)	/users/{id}/orders/{oid}	/a/b/c/d/e/f

CH 04

HTTP 메서드와 의미

행위(Action)는 URI 가 아닌 메서드로 표현한다

4.1 메서드 개요

메서드	역할	멱등성	안전성	주요 용도
GET	자원 조회 (Read)	✅ 예	✅ 예	목록/단건 조회
POST	생성 / 액션 (Create)	❌ 아니요	❌ 아니요	생성, 비동기 요청
PUT	전체 대체 (Replace)	✅ 예	❌ 아니요	전체 필드 교체
PATCH	부분 수정 (Update)	△ 조건부	❌ 아니요	일부 필드 수정
DELETE	삭제 (Delete)	✅ 예	❌ 아니요	자원 제거

4.2 각 메서드 상세

◆ GET — 조회

서버 상태를 바꾸지 않아야 하며, 같은 요청은 같은 의미를 가져야 한다. GET 요청에 본문(body)을 기대하는 설계는 피한다.

◆ POST — 생성 / 액션

새로운 자원 생성, 비동기 작업 요청, 명시적 액션 실행에 사용. POST 는 일반적으로 멱등하지 않으므로, 결제 요청 등에는 멱등성 키를 별도 도입한다.

◆ PUT vs PATCH

```
# PUT — 전체 교체: 누락된 필드는 제거됨
PUT /users/123 { "name":"홍길동","email":"hong@example.com","phone":"..." }

# PATCH — 부분 수정: 보낸 필드만 업데이트
PATCH /users/123 { "phone": "010-9999-0000" }
```

◆ DELETE — 삭제

실제로는 소프트 삭제일 수 있지만, 외부 계약상 삭제로 표현해도 무방하다. 삭제 후 조회 응답, 복구 가능 여부, 참조 관계 처리를 사전에 설계해야 한다.

CH 05

요청과 응답 모델 설계

일관되고 예측 가능한 페이로드 구조

5.1 요청 모델 설계 원칙

- 필수와 선택을 명확히 나눈다
- 각 필드의 단위와 형식을 명시한다
- 검증 규칙을 문서화한다
- 서버가 무시할 필드는 받지 않는다

5.2 응답 모델 설계 원칙

응답은 화면 구현에 맞춘 임시 구조가 아니라 도메인 의미 중심으로 설계해야 한다. 프론트엔드 편의를 위해 화면 종속적으로 만들면 다른 소비자가 재사용하기 어렵다.

5.3 JSON 필드 네이밍

camelCase 또는 snake_case 중 하나를 조직 표준으로 고정한다. 중요한 것은 취향이 아니라 일관성이다.

```
// ✅ camelCase
{ "userId": "usr_123", "createdAt": "2025-03-25T09:00:00Z", "isActive": true }

// ❌ 혼용 금지
{ "user_id": "usr_123", "createdAt": "..." }
```

5.4 날짜와 시간

ISO 8601 / RFC 3339 기반 UTC 시각으로 표준화한다. 예: 2025-03-25T09:00:00Z

5.5 null, 빈 문자열, 필드 누락의 의미

표현	의미	PATCH 에서의 해석
null	값이 명시적으로 비어 있음	해당 필드를 제거(비움)
"" (빈 문자열)	빈 문자열이라는 값 자체	빈 문자열로 설정
필드 누락	제공되지 않음	변경하지 않음 (유지)

CH 06

상태 코드와 에러 모델

클라이언트는 상태 코드만 보고도 결과를 알 수 있어야 한다

6.1 HTTP 상태 코드를 존중하라

상태 코드는 단순 장식이 아니다. 클라이언트, 프록시, 로깅, APM, 모니터링, 재시도 로직이 모두 상태 코드에 의존한다.

코드	이름	사용 시점
200 OK	OK	정상 조회/수정
201 Created	Created	자원 생성 성공
202 Accepted	Accepted	비동기 처리 접수
204 No Content	No Content	본문 없는 성공 (DELETE)
400 Bad Request	Bad Request	잘못된 요청 형식
401 Unauthorized	Unauthorized	인증 필요 또는 인증 실패
403 Forbidden	Forbidden	인증됐으나 권한 부족
404 Not Found	Not Found	자원 없음
409 Conflict	Conflict	상태 충돌, 중복 생성
422 Unprocessable	Unprocessable Entity	비즈니스 검증 실패
429 Too Many Requests	Too Many Requests	호출 제한 초과
500 Internal Error	Internal Server Error	서버 내부 오류
503 Service Unavailable	Service Unavailable	일시적 장애/점검

6.2 공통 에러 응답 형식

표준 에러 응답 구조

```
{
  "error": {
    "code": "INVALID_PARAMETER",
    "message": "nickname must be between 2 and 20 characters",
    "details": [{ "field": "nickname", "reason": "length_out_of_range" }],
    "requestId": "req_01HQX..."
  }
}
```

6.3 내부 예외를 그대로 노출하지 말 것

보안 경고
스택 트레이스, SQL 에러, 내부 클래스명은 외부 응답에 절대 노출하지 않는다. 내부 로그에 상세히 남기되, 외부에는 통제된 에러 메시지만 제공한다.

6.4 검증 오류 설계

검증 실패 응답 예시

```
{
  "error": {
    "code": "INVALID_PARAMETER",
    "message": "One or more fields are invalid.",
    "details": [
      { "field": "email",  "reason": "invalid_format" },
      { "field": "age",    "reason": "must_be_greater_than_or_equal_to_14" }
    ]
  }
}
```

CH 07

인증과 권한

Authentication & Authorization

7.1 인증과 권한을 분리해서 생각하라

구분	정의	HTTP 상태 코드
인증 (Authentication)	누구인지 확인하는 과정	401 Unauthorized
권한 (Authorization)	무엇을 할 수 있는지 결정	403 Forbidden

7.2 인증 방식 선택

방식	특징	적합한 사용처	주의사항
세션/쿠키	서버 저장 상태	전통적 웹앱	수평 확장 어려움
Bearer(JWT)	자체 포함 토큰, Stateless	마이크로서비스, SPA	만료 시간 필수
API Key	헤더에 포함, 단순	서버 간 통신	주기적 교체 필요
OAuth 2.0	위임 권한 부여 표준	소셜 로그인, 3rd Party	구현 복잡

7.3 권한 모델 설계

- 역할 기반 권한 (RBAC)
- 자원 소유권 기반 권한
- 조직/테넌트 범위 권한
- 상태 기반 권한
- 세부 스코프 기반 권한

7.4 민감정보 최소화

- 가능한 저장하지 않는다
- 저장한다면 암호화/마스킹/접근 통제를 적용한다
- 로그와 에러 응답에 포함하지 않는다

🔒 **응답 필드 점검**
사용자 조회 응답에 비밀번호 해시, 주민등록번호, 원본 토큰이 실수로 포함되지 않도록 주의한다.

CH 08

조회 API 설계

필터링, 정렬, 페이지네이션

8.1 목록 조회의 표준화

목록 조회 API는 대부분 서비스에서 가장 많이 호출된다. 규칙이 제각각이면 프론트엔드와 운영도구 생산성이 크게 떨어진다.

8.2 필터링

```
GET /orders?status=paid&from=2025-03-01T00:00:00Z&to=2025-03-31T23:59:59Z
```

8.3 정렬

```
GET /products?sort=-createdAt,name    # - 는 내림차순
```

8.4 페이지네이션

방식	파라미터	특징
오프셋 기반	page, size	구현 쉬움. 대규모에서 성능 저하
커서 기반	cursor, limit	무한 스크롤에 유리. 구현 복잡

8.5 목록 응답 구조

```
{
  "items": [{ "orderId": "ord_001", "status": "paid", "createdAt": "2025-03-
25T09:00:00Z" }],
  "pageInfo": { "nextCursor": "abc123", "hasNext": true }
}
```

CH 09

생성 / 수정 / 삭제 API 설계

CUD API 설계 가이드

9.1 생성 API

201 상태와 함께 생성된 자원의 식별자 또는 전체 표현을 반환한다. Location 헤더에 생성된 자원 URL 포함 권장.

9.2 수정 API

- 부분 수정인지 전체 대체인지
- 수정 불가능한 필드 정의
- 상태 전이 제약 설계

9.3 삭제 API

- 하드 vs 소프트 삭제 결정
- 연관 자원 처리 방식
- 복구 가능 여부

CH 10

멱등성, 재시도, 중복 방지

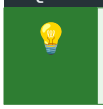
Idempotency & Retry

10.1 멱등성의 중요성

타임아웃, 중복 클릭, 재전송, 모바일 네트워크 불안정은 흔하다. 생성/결제/예약 API 는 중복 처리 방지를 반드시 고려해야 한다.

10.2 Idempotency-Key 패턴

```
POST /payments
Idempotency-Key: 6b8d1f8c-ae20-4b7d-8f3e-1c2d3e4f5a6b
{ "amount": 32000, "currency": "KRW" }
```



적용 대상

결제, 주문 생성, 메시지 발송 예약 등 중복 처리가 치명적인 작업에 필수.

CH 11

비동기 처리와 장기 작업 API

Long-Running Task Pattern

11.1 비동기 API 패턴

```
POST /exports → 202 Accepted + { "exportId": "exp_001", "status": "queued" }
GET /exports/exp_001 → { "status": "processing", "progress": 45 }
                     → { "status": "completed", "result": { "downloadUrl": "..." }
}
```

11.2 작업 상태 모델

상태	설명	다음 상태
queued	작업 큐에 등록됨	processing
processing	처리 중	completed / failed
completed	처리 완료	(최종 상태)
failed	처리 실패	재시도 또는 canceled

CH 12

버전 관리와 하위 호환성

API 진화 전략

12.1 버전이 필요한 순간

- 응답 구조 대폭 변경
- 필수 요청값 추가
- 필드 의미 변경
- 인증 방식 변경

12.2 하위 호환성 원칙

- 기존 필드를 삭제하지 않는다
- 기존 필드 타입을 바꾸지 않는다
- 선택 필드는 추가 가능하나 소비자가 무시해도 동작해야 한다

12.3 Deprecated 전략

- 문서에 deprecated 표기
- 대체 수단 안내
- 최소 6 개월 유예 기간
- Sunset 헤더 활용

CH 13

문서화와 API 계약 관리

Documentation as a Contract

13.1 문서는 설계의 일부다

문서 없는 API 는 존재하지 않는 것과 비슷하다. 모든 엔드포인트의 요청/응답 예시, 에러 코드, 인증 방식, 변경 이력을 포함하라.

13.2 OpenAPI 활용

OpenAPI(Swagger)는 문서화뿐 아니라 계약 검증, 코드 생성, 테스트 자동화에도 유용하다.

⚠ **가장 위험한 상태**
"문서는 있는데 틀린 문서"다. CI 에서 OpenAPI 스펙 검증과 계약 테스트를 자동화하라.

CH 14

보안 설계

Security by Design

14.1 입력 검증

모든 외부 입력은 신뢰하지 않는다. 문자열 길이, 형식, 범위, enum 값, 파일 크기를 검증하라.

14.2 흔한 보안 고려사항

- SQL/NoSQL Injection 방지
- XSS 방지
- CSRF 대응
- CORS 최소 허용
- Rate Limiting
- Brute force 대응
- 파일 업로드 검사
- SSRF 방지
- 민감정보 마스킹
- 감사 로그

⚠ **IDOR 공격 주의**
다른 사용자의 자원을 ID 만 바꿔 조회하는 IDOR(Insecure Direct Object Reference) 공격을 반드시 방지해야 한다.

CH 15

관측성, 로그, 추적성

Observability

15.1 requestId 와 correlationId

모든 요청에 추적 가능한 ID 를 부여한다. 응답 헤더와 에러 본문에 requestId 를 포함하면 운영 분석에 큰 도움이 된다.

15.2 구조화 로그

requestId, userId, endpoint, statusCode, latencyMs, errorCode, tenantId 를 키-값 형태로 로깅한다.
민감정보는 로그에 남기지 않는다.

15.3 핵심 지표

- 요청 수 (QPS)
- 응답 시간 (p50, p95, p99)
- 에러율
- 상태 코드 분포
- Rate Limit 초과율

CH 16

API 테스트 전략

Test Strategy

테스트 종류	목적	도구 예시
단위 테스트	검증 로직, 매퍼 검증	JUnit, Jest, pytest
통합 테스트	DB/캐시 포함 실제 흐름	Testcontainers, Supertest
계약 테스트	요청/응답 스키마 일치	Pact, Dredd
부하 테스트	병목과 한계 확인	k6, Gatling, JMeter
보안 테스트	인증/권한 우회 시도	OWASP ZAP, Burp Suite

CH 17

팀 공통 스타일 가이드

Style Guide

17.1 URL 규칙

- 명사형 자원명
- 복수형 기본
- 소문자와 하이픈
- 동사형 URL 지양
- 깊이 3 단계 이내

17.2 JSON 규칙

- 필드명 camelCase 통일
- 날짜 UTC ISO 8601
- boolean: is/has/can 접두어
- 식별자는 문자열

17.3 상태 코드 규칙

- HTTP 상태 코드에 성공/실패 반영
- 200 남용 지양
- 검증 실패(400/422)와 권한 실패(401/403) 구분

17.4 에러 규칙

- 공통 포맷
- code 안정적 유지
- requestId 포함

17.5 문서 규칙

- 모든 공개 API 는 OpenAPI 스펙
- 성공 1 개, 실패 2 개 이상 예시
- 변경 이력
- deprecated: 제거 일정 + 대체 경로

CH 18

실전 예시: 주문 API 설계

지금까지의 원칙을 실제 API 로

18.1 주문 생성

```
POST /v1/orders
{ "items":[{"productId":"prd_001","quantity":2}],
"shippingAddressId":"addr_123" }

→ 201 Created
{ "orderId":"ord_001","status":"pending","totalAmount":
{"currency":"KRW","value":32000},"createdAt":"2025-03-25T09:00:00Z" }
```

18.2 주문 목록 조회

```
GET /v1/orders?status=paid&cursor=abc123&limit=20&sort=-createdAt

{ "items":[{"orderId":"ord_001","status":"paid","totalAmount":
{"currency":"KRW","value":32000}}],
  "pageInfo":{"nextCursor":"def456","hasNext":true} }
```

18.3 주문 취소

```
POST /v1/orders/{orderId}/cancel
{ "reason": "changed_mind" }

// 실패 (422)
{ "error":{ "code":"INVALID_ORDER_STATE","message":"Paid orders cannot change
address.","requestId":"req_123" } }
```

CH 19

안티패턴 모음

하지 말아야 할 것들

19.1 항상 200 반환

나쁜 예

HTTP/1.1 200 OK {"success":false,"code":"USER_NOT_FOUND"} ← 404 를 써야 한다

19.2 DB 스키마 직노출

테이블 컬럼을 그대로 외부 API 에 노출하면 내부 리팩터링이 어려워진다.

19.3 화면 맞춤형 과도한 응답

특정 화면 하나를 위한 맞춤형 구조는 다른 소비자 재사용성을 떨어뜨린다.

19.4 불분명한 상태값

A, B, C 같은 약어 대신 pending, paid, canceled 처럼 의미가 드러나야 한다.

19.5 페이징 없는 대량 목록

페이지네이션 없이 전체 데이터를 반환하면 운 좋을 때만 동작한다.

CH 20

설계 리뷰 체크리스트

API 출시 전 최종 점검

20.1 기본 구조

	확인 항목
☐	이 API 의 소비자는 누구인가?
☐	도메인 자원이 명확한가?
☐	URL 이 자원 중심으로 설계되었는가?
☐	HTTP 메서드 의미가 적절한가?

20.2 요청/응답

	확인 항목
☐	필수/선택 필드가 명확한가?
☐	날짜·금액·enum 형식이 일관적인가?
☐	null/누락/빈값 의미가 정의되었는가?
☐	응답이 화면 종속적이지 않은가?

20.3 에러/상태 코드

	확인 항목
☐	상태 코드가 적절한가?
☐	공통 에러 포맷을 따르는가?
☐	검증 오류가 필드 단위로 설명되는가?
☐	내부 예외가 노출되지 않는가?

20.4 보안/권한

	확인 항목
☐	인증 실패(401)와 권한 실패(403)가 구분되는가?
☐	소유권/테넌트 경계가 고려되었는가?
☐	민감정보가 노출되지 않는가?
☐	Rate limiting 이 필요한가?

20.5 운영/변경 관리

	확인 항목
☐	requestId 와 로그 전략이 있는가?
☐	재시도와 멱등성이 필요한가?
☐	OpenAPI 문서가 준비되었는가?
☐	하위 호환성 영향이 검토되었는가?

CH 21

조직 도입 전략

How to Adopt This Guide

21.1 도입 우선순위

순서	도입 항목	이유
1	공통 에러 포맷	모든 API 에 즉시 적용, 가시 효과 큼
2	상태 코드 기준	클라이언트 협업 즉시 개선
3	페이지네이션 규칙	목록 API 일관성
4	인증 방식	보안 기반 통일
5	OpenAPI 문서화	설계 계약 가시화
6	requestId / 로깅	운영 추적 가능성

21.2 신규 API 부터 적용

기존 API 전체를 한 번에 뜯어고치기보다 신규 API 에 우선 적용한다. 이후 린트, 템플릿, 리뷰 체크리스트로 확산시킨다.

21.3 설계 리뷰 문화

좋은 API 품질은 개인 역량보다 리뷰 문화에서 나온다. PR 이전에 API 설계 리뷰를 별도로 두면 구현 후 되돌리는 비용을 크게 줄인다.

리뷰 타이밍
구현 전 설계 리뷰 → 구현 후 계약 테스트 → 출시 전 체크리스트. 이 세 단계를 팀 프로세스에 포함하라.

맺음말

백엔드 API 설계는 단순히 URL 과 JSON 을 정하는 일이 아니다. 서비스의 경계를 정의하고, 팀의 협업 방식을 결정하며, 운영 안정성과 확장성을 좌우하는 핵심 설계 행위다.

> *"설계는 구현보다 오래 간다. 그러므로 API 는 코드를 작성하기 전에 먼저 신중하게 설계되어야 한다."*

부록

부록 A. 추천 공통 에러 코드

에러 코드	설명
INVALID_PARAMETER	입력값 형식 또는 범위 오류
INVALID_STATE	현재 상태에서 허용되지 않는 작업
UNAUTHORIZED	인증 필요 또는 인증 실패
FORBIDDEN	인증됐지만 권한 없음
RESOURCE_NOT_FOUND	요청한 자원이 존재하지 않음
ALREADY_EXISTS	중복 자원 생성 시도
RATE_LIMIT_EXCEEDED	호출 제한 초과
CONFLICT	동시성 또는 상태 충돌
INTERNAL_ERROR	서버 내부 오류
DEPENDENCY_FAILURE	외부 서비스 의존성 오류

부록 B. 추천 공통 응답 메타 헤더

헤더	용도
X-Request-Id	요청 추적 ID
X-RateLimit-Limit / Remaining / Reset	Rate Limit 정보
Location	생성된 자원 URL (201 시)
Retry-After	재시도 가능 시간 (429, 503 시)
Deprecation / Sunset	API 비권장 및 제거 예정일

부록 C. 한 페이지 요약

#	핵심 원칙
1	자원 중심으로 설계하라
2	HTTP 의미와 상태 코드를 존중하라
3	요청/응답/에러 형식을 일관되게 유지하라
4	인증, 권한, 민감정보 보호를 처음부터 고려하라
5	문서, 버전, 변경 관리를 계약의 일부로 보라

#	핵심 원칙
6	로그, 추적, 재시도, 멱등성까지 포함해 운영 가능하게 설계하라

백엔드 API 설계를 위한 실무 지침서 v3.0 — END