

# WBX Spring Framework

## 개발자 가이드

Developer Guide v2.0 (Spring Boot 3.5)

아큐라시스템 | 2026년 3월

# 목차

- Chapter 0. 개발환경 설정 — IDE · 프로젝트 생성
- Chapter 1. Quick Start — 5분 만에 시작하기
- Chapter 2. 프로젝트 구조 · Gradle 설정
- Chapter 3. 인증 — JWT · SSO · MFA
- Chapter 4. 권한 — RBAC · dept\_scope
- Chapter 5. 결재 — Handler 구현 가이드
- Chapter 6. 알림 — SSE · WebSocket
- Chapter 7. WBX No-Code 플랫폼 연동
- Chapter 8. DB 설정 — Multi DBMS · DataSource
- Chapter 9. API 응답 규칙 · 에러 처리
- Chapter 10. 파일 업로드 · 스토리지
- Chapter 11. 테스트 가이드
- Chapter 12. 배포 — 클라우드 · CI/CD
- Chapter 13. Admin UI · 시스템 관리
- Chapter A. 설정 레퍼런스 (application.yml)

## Chapter 0. 개발환경 설정 - IDE · 프로젝트 생성

### 0-1. 사전 준비

- JDK 21 설치 (Eclipse Temurin 권장)
- Git 설치

**NOTE:** Redis는 Embedded Redis가 앱과 함께 자동 시작되므로 별도 설치가 필요 없습니다. Docker Desktop도 DB를 직접 설치하면 불필요합니다.

### 온보딩 플로우

1) JDK 설치 -> 2) IDE 설정 -> 3) DB 준비 -> 4) 프로젝트 생성 -> 5) bootRun -> 6) Swagger 확인

### 0-2. IDE 선택 가이드

IDE	권장 대상	핵심 장점
IntelliJ IDEA	메인 개발	Spring Boot 최적, 리팩터링, DB 브라우저
VS Code	경량/FE 병행	Extension Pack, DevContainer
Eclipse/STS	무료/레거시	Spring Tool Suite 플러그인

### 0-3. IntelliJ IDEA 필수 설정

- 플러그인: Spring Boot, Lombok, JPA Buddy, GitToolBox, .env, SonarLint
- Annot

Run Configuration:  
Main class: kr.co.accura.wbx.spring.WbxSpringCoreApplication  
Env: JWT\_SECRET=dev-secret-key

### 0-4. VS Code 필수 설정

- Extension: Java Pack, Spring Boot Pack, Lombok, Gradle, REST Client, GitLens
- .VSCO

```
// .vscode/launch.json
{
  "type": "java",
  "name": "App (Local)",
  "mainClass": "kr.co.accura.wbx.spring.WbxSpringCoreApplication",
  "env": {"JWT_SECRET": "dev-secret-key"}
}
```

## 0-5. Eclipse / STS 설정

- Spring Tools 4 플러그인 설치 (Eclipse Marketplace)

- Lombok

## 0-6. 프로젝트 생성 (3가지 방법)

방법 A: Spring Initializr (start.spring.io)

```
Project: Gradle-Kotlin | Java: 21 | Boot: 3.5.0
Group: kr.co.accura | Artifact: {앱}-api
-> Generate -> 압축 해제 -> wbx-spring-starter 의존성 추가
```

방법 B: Git Template Repository

```
git clone https://git.wbx.kr/accura/wbx-spring-template.git my-app
cd my-app && ./init.sh --name my-app --group kr.co.accura
```

## 0-7. 설치 스크립트 (권장)

프로젝트에 포함된 설치 스크립트가 JDK 자동 설치, Git 사전 검사, 빌드, .env 템플릿 생성을 자동 처리합니다.

```
# Windows
scripts\install.bat

# Linux/macOS
chmod +x scripts/install.sh && ./scripts/install.sh
```

TIP: JDK 미설치 시 스크립트가 자동으로 설치합니다.

## 0-8. 로컬 개발 인프라

Redis는 Embedded Redis가 앱 시작 시 자동 구동되므로 별도 설치가 필요 없습니다.

DB만 Docker Compose 또는 직접 설치하면 됩니다.

```
# Docker로 DB만 시작 (Redis 불필요)
docker compose -f docker-compose-dev.yml up -d mysql

# 또는 DB를 직접 설치한 경우 바로 앱 실행
gradlew.bat bootRun
```

**Embedded Redis 동작:** 앱 시작 시 외부 Redis 감지 -> 없으면 자동 시작 -> 실패 시 인메모리 캐시

## 0-9. DevContainer (VS Code)

devcontainer.json 으로 JDK 21 + DB가 포함된 완전한 개발 환경을 컨테이너로 제공합니다.  
Redis는 Embedded Redis가 자동 구동되므로 DevContainer에 별도 포함하지 않아도 됩니다.  
팀원 전체 동일 환경 보장.

## 0-10. 코드 품질 도구

- Spotless - 코드 포맷 자동화 (./gradlew spotlessApply)

- Check

# Chapter 1. Quick Start — 5분 만에 시작하기

## 1-1. 의존성 추가

```
// build.gradle.kts
dependencies {
    implementation("kr.co.accura:wbx-spring-starter:1.0.0")
}
```

## 1-2. application.yml 설정

```
wbx:
  spring:
    api-prefix: /api/myapp    # API URL prefix
  jwt:
    secret: ${JWT_SECRET}    # 환경변수
    expiration: 28800        # 8시간
  approval:
    enabled: true
  notification:
    sse-enabled: true
```

## 1-3. 첫 번째 Controller

```
@RestController
@RequestMapping("${wbx.spring.api-prefix}/v1/hello")
public class HelloController {

    @GetMapping
    @PreAuthorize("isAuthenticated()")
    public Map<String, String> hello() {
        return Map.of("message", "Hello WBX Spring!");
    }
}
```

## 1-4. 실행

```
# 로컬 실행
./gradlew bootRun

# 확인 (기본 context-path: /spring)
http://localhost:8080/spring/health
http://localhost:8080/spring/swagger-ui
http://localhost:8080/spring/admin/login
```

**TIP:** wbx-spring-starter가 JWT, CORS, 에러 핸들링, 페이징 변환을 자동 설정합니다.

## Chapter 2. 프로젝트 구조 · Gradle 설정

### 2-1. 권장 패키지 구조

```
kr.co.accura.{app}/
├── domain/           # 비즈니스 도메인
│   ├── {module}/
│   │   ├── entity/
│   │   ├── repository/
│   │   ├── service/
│   │   └── dto/
│   └── approval/
│       └── handler/ # 결재 핸들러만
├── api/              # REST Controller
├── integration/      # 외부 연동
└── config/           # 앱 전용 설정
```

### 2-2. 네이밍 규칙

항목	패턴	예시
API Prefix	/api/{약칭}/v1	/api/wtm/v1
Java 패키지	kr.co.accura.{약칭}	kr.co.accura.wtm
Gradle artifact	{약칭}-api	wtm-api
DB	{약칭}_db	wtm_db
Entity	단수 PascalCase	Timesheet
Controller	도메인+Controller	TimesheetController
목록 응답 키	items	{"items":[...], "total":150}
에러 응답 키	detail	{"detail":"에러 메시지"}

## Chapter 3. 인증 — JWT · SSO · MFA

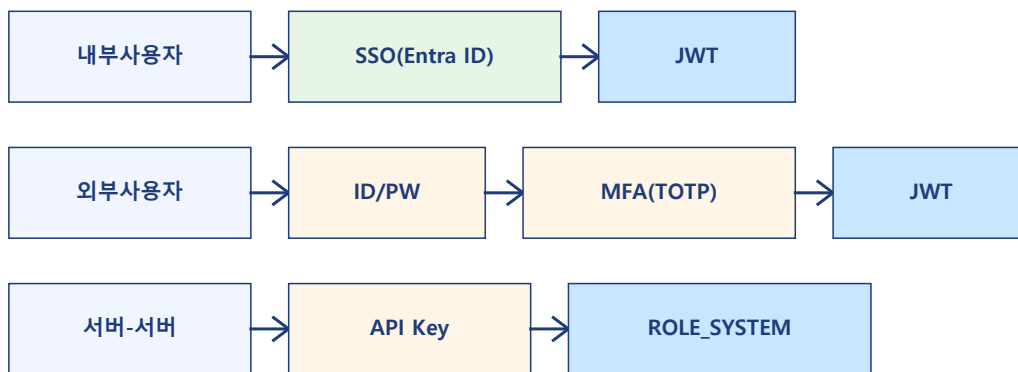
### 3-1. 인증 흐름

내부 사용자: Azure Entra ID SSO → JWT 발급 (Entra Conditional Access로 MFA)

외부 사용자: ID/PW → MFA TOTP 검증 → JWT 발급

서버-서버: X-API-Key 헤더 → ROLE\_SYSTEM 인증

#### 인증 흐름도



### 3-2. JWT Payload (WBX 호환)

```

{
  "sub": "user@company.com", // WBX 호환: email
  "user_id": 42,             // WBX 호환: 사용자 ID
  "username": "hong",
  "full_name": "홍길동",
  "email": "user@company.com",
  "is_admin": false,
  "department_id": 3,
  "roles": ["PM", "DL"],
  "iat": 1711929600,
  "exp": 1711958400
}
  
```

**주의:** SECRET\_KEY를 WBX FastAPI와 동일하게 설정해야 SSO가 작동합니다.

### 3-3. MFA 설정

```

wbx:
  spring:
    mfa:
      enabled: true
  
```

```

force-for-external: true # 외부사용자 필수
force-for-internal: false # 내부=Entra CA
totp-issuer: "WBX Platform"
totp-digits: 6
totp-period: 30

```

### 3-4. MFA API 엔드포인트

Path	Method	설명
/auth/login	POST	1단계: ID/PW → JWT 또는 mfa_required
/auth/mfa/verify	POST	2단계: TOTP 코드 → JWT 발급
/auth/mfa/setup	POST	MFA 설정 시작 (QR otpauth URI)
/auth/mfa/setup/verify	POST	MFA 활성화 + 백업코드 발급
/auth/mfa	DELETE	MFA 비활성화 (본인/SA)
/auth/mfa/backup-verify	POST	백업코드로 인증 (1회용)

### 3-5. PasswordPolicy (비밀번호 정책)

PasswordPolicy 클래스가 비밀번호 규칙을 자동 검증합니다.

```

// auth/PasswordPolicy.java
@Component
public class PasswordPolicy {
    // application.yml에서 설정
    // wbx.spring.password.min-length: 8
    // wbx.spring.password.require-uppercase: true
    // wbx.spring.password.require-digit: true
    // wbx.spring.password.require-special: true
    // wbx.spring.password.max-failed-attempts: 5
    // wbx.spring.password.expiry-days: 90

    public void validate(String password) { ... }
    public boolean isExpired(WbxUser user) { ... }
    public boolean isLocked(WbxUser user) { ... }
}

```

### 3-6. LoginHistory (로그인 이력)

로그인 성공/실패/로그아웃이 자동으로 wbx\_login\_history 테이블에 기록됩니다.

Admin Console → 로그인 이력에서 조회 가능합니다.

```

// 자동 기록되는 항목:
// action: LOGIN_SUCCESS, LOGIN_FAILURE, LOGOUT
// auth_method: PASSWORD, SSO, API_KEY, REFRESH_TOKEN
// ip_address, user_agent 자동 수집

```

### 3-7. AuditLogService (감사 로그)

비즈니스 로직에서 수동으로 감사 로그를 기록합니다.

```
// 서비스에서 사용
@Autowired AuditLogService auditLog;

auditLog.log("UPDATE", "TIMESHEET", timesheetId, "시수 수정");
// → wbx_audit_logs 테이블에 저장
// → userId, username, ipAddress 자동 수집 (SecurityUtils)
```

## Chapter 4. 권한 — RBAC · dept\_scope

---

### 4-1. Controller에서 권한 체크

```
// 모듈-액션 기반 체크
@GetMapping("/timesheets")
@PreAuthorize("@wbx.check('TIMESHEET', 'VIEW')")
public Map<String, Object> list() { ... }

// dept_scope로 데이터 필터링
DeptScope scope = evaluator.getScope("TIMESHEET", "VIEW");
// OWN → 본인만, DEPT → 부서, COMPANY → 전사
```

### 4-2. dept\_scope 자동 필터링

```
Specification<Timesheet> spec = scope.toSpec(
    userId, deptId, "userId", "departmentId");
Page<TimesheetDto> page = repo.findAll(spec, pageable);
```

## Chapter 5. 결재 — Handler 구현 가이드

### 5-1. 결재 핸들러 등록 (3단계)

1. ApprovalHandler 인터페이스 구현
2. @Component 어노테이션 추가
3. 끝 — Registry가 자동 수집, API 자동 활성화

#### Handler 등록 흐름



결재 흐름 예시:



### 5-2. 핸들러 구현 예시

```

@Component
public class TimesheetApprovalHandler implements ApprovalHandler {

    @Override
    public String getTypeKey() { return "timesheet"; }

    @Override
    public String getTypeDisplay() { return "시수 결재"; }

    @Override @Transactional
    public ApprovalResult approve(Long lineId, Long approverId,
                                  String comment) {
        // 비즈니스 로직 구현
        return ApprovalResult.success("승인 완료");
    }

    // reject(), getApprovalHistory(), getPending() ...
}
  
```

### 5-3. 자동 생성되는 API

```

POST /api/{prefix}/approvals/unified/action/timesheet/{id}/approve
POST /api/{prefix}/approvals/unified/action/timesheet/{id}/reject
GET  /api/{prefix}/approvals/unified/pending
  
```

TIP: `getTypeKey()`만 다르면 여러 결재 유형을 동시에 운영할 수 있습니다.

## Chapter 6. 알림 — SSE · WebSocket

---

### 6-1. 알림 전송 (서비스에서)

```
// wbx-spring이 제공하는 SseNotificationService 주입
@Autowired
private SseNotificationService notificationService;

// 특정 사용자에게 알림
notificationService.sendToUser(approverId,
    NotificationDto.builder()
        .type("APPROVAL_REQUEST")
        .title("결재 요청")
        .message("홍길동님의 시수 결재")
        .link("/approvals/123")
        .build());
```

## Chapter 7. WBX No-Code 플랫폼 연동

---

### 7-1. API 등록 (apis 테이블)

```
INSERT INTO apis (project_id, name, method, endpoint, response_path)
VALUES (166, '시수 목록', 'GET', '/api/wtm/v1/timesheets', 'items');
```

### 7-2. 응답 규칙

- 목록: response\_path='items' → {"items":[...], "total":150}
- 단일: response\_path="" → 객체 직접 반환
- 에러: {"detail":"에러 메시지"}

**주의:** DELETE 응답은 204가 아닌 200 + body로 반환해야 WBX toast가 작동합니다.

## Chapter 8. DB 설정 — Multi DBMS

### 8-1. 프로필로 DB 전환

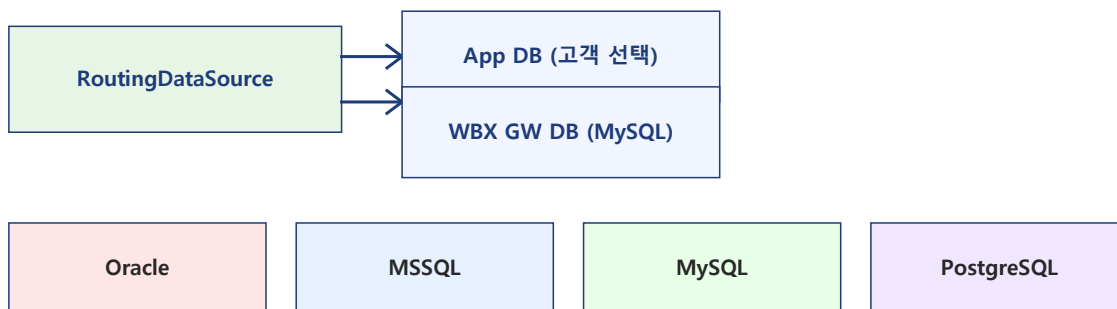
```
# Oracle
java -jar app.jar --spring.profiles.active=prod,oracle

# MSSQL (Azure SQL)
java -jar app.jar --spring.profiles.active=prod,mssql

# MySQL
java -jar app.jar --spring.profiles.active=prod,mysql

# PostgreSQL
java -jar app.jar --spring.profiles.active=prod,postgresql
```

### Multi-DataSource 구조



App DB: 프로필로 DBMS 선택 (Oracle / MSSQL / MySQL / PostgreSQL)

### 8-2. Entity 작성 규칙 (DBMS 독립)

- columnDefinition 사용 금지 (DBMS 종속)
- @GeneratedValue(strategy = IDENTITY) 사용
- Boolean → Java boolean (Hibernate 자동 변환)
- 날짜 → LocalDateTime / LocalDate (자동 매핑)
- 대용량 텍스트 → @Lob 대신 length 지정

### 8-3. @DataSource AOP (Multi-DataSource)

```
# 활성화 (application.yml)
wbx.spring.datasource.routing-enabled: true

# WBX GW DB 연결 설정
wbx.spring.datasource.wbxgw:
  url: jdbc:mysql://localhost:3306/wbx_gw
  username: wbxapp
```

```
password: password
```

```
// 서비스에서 @DataSource로 DB 전환
@DataSource("wbxgw") // WBX GW DB에서 읽기
public List<GwUser> getGwUsers() { ... }

@DataSource("app") // 기본값, 생략 가능
public void saveUser(User user) { ... }
```

## 8-4. Flyway 마이그레이션

4종 DBMS별 DDL이 준비되어 있습니다. 프로필 전환 시 자동 적용.

```
src/main/resources/db/migration/
├── common/      # 공통 Seed (INSERT)
├── mysql/       # MySQL DDL (11 테이블)
├── postgresql/  # PostgreSQL DDL
├── oracle/      # Oracle DDL
└── mssql/       # MSSQL DDL
```

**TIP: 개발 시 `hibernate.ddl-auto=update`, 프로덕션 시 `flyway.enabled=true + ddl-auto=validate`**

## Chapter 9. API 응답 규칙 · 에러 처리

---

### 9-1. 표준 Controller 패턴

```
// 목록
@GetMapping
public Map<String, Object> list(
    @RequestParam(defaultValue = "0") int skip,
    @RequestParam(defaultValue = "20") int limit) {
    Page<Dto> page = service.findAll(...);
    return Map.of("items", page.getContent(),
        "total", page.getTotalElements());
}

// 단일
@GetMapping("/{id}")
public Dto getOne(@PathVariable Long id) {
    return service.findById(id);
}

// 에러는 자동 처리 (WbxErrorHandler)
// BusinessException → {"detail": "..."} (400)
// NotFoundException → {"detail": "..."} (404)
```

## Chapter 10. 파일 업로드 · 스토리지

### 10-1. 스토리지 전환

```
wbx:
  spring:
    file:
      storage-type: azure-blob # aws-s3 | gcp-storage | local
```

FileStorageService 인터페이스를 통해 코드 변경 없이 스토리지를 전환합니다.  
upload(), download(), delete(), getPresignedUrl() 메서드를 동일하게 사용합니다.

### 10-2. 클라우드 스토리지 설정

```
# Azure Blob
wbx.spring.file.storage-type: azure-blob
wbx.spring.file.azure:
  account-name: ${AZURE_STORAGE_ACCOUNT}
  account-key: ${AZURE_STORAGE_KEY}
  container-name: uploads

# AWS S3
wbx.spring.file.storage-type: aws-s3
wbx.spring.file.aws:
  bucket: ${AWS_S3_BUCKET}
  region: ap-northeast-2

# GCP Cloud Storage
wbx.spring.file.storage-type: gcp-storage
wbx.spring.file.gcp:
  bucket: ${GCP_STORAGE_BUCKET}
```

### 10-3. 서비스에서 파일 업로드

```
@Autowired FileStorageService fileStorage;

// 업로드
String key = fileStorage.upload(multipartFile, "documents");

// 다운로드
byte[] data = fileStorage.download(key);

// 임시 URL (클라우드)
String url = fileStorage.getPresignedUrl(key, 3600);
```

**TIP:** 프로덕션은 클라우드 SDK로 교체를 권장합니다: `spring-cloud-azure-starter-storage-blob`, `spring-cloud-aws-starter-s3`

## Chapter 11. 테스트 가이드

---

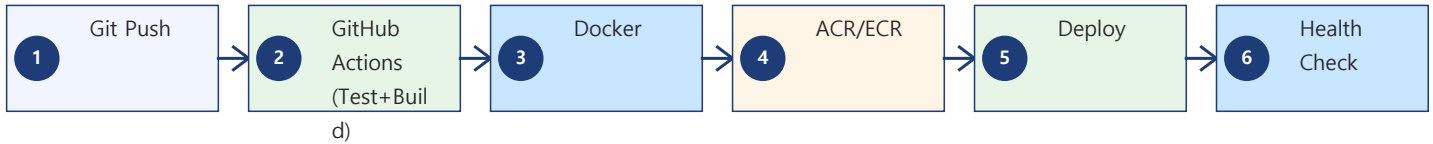
```
# application-test.yml
spring:
  datasource:
    app:
      url: jdbc:h2:mem:testdb;MODE=PostgreSQL
  jpa:
    hibernate:
      ddl-auto: create-drop
  flyway:
    enabled: false
```

```
@SpringBootTest
class TimesheetServiceTest {
    @Autowired TimesheetService service;

    @Test
    void 주간시수_생성() {
        var ts = service.getOrCreate(userId, weekStart);
        assertThat(ts.getStatus()).isEqualTo(DRAFT);
    }
}
```

## Chapter 12. 배포 — 클라우드 · CI/CD

### CI/CD 파이프라인



### 12-1. 프로필 조합

```

# 환경 + 클라우드 + DB 조합
--spring.profiles.active=prod,azure,mssql
--spring.profiles.active=prod,aws,postgresql
--spring.profiles.active=prod,onprem,oracle
  
```

### 12-2. Docker

```

FROM eclipse-temurin:21-jre-alpine
COPY build/libs/app.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
  
```

### 12-3. Kubernetes Helm

```

helm install myapp ./helm/app -f values-azure.yaml
helm install myapp ./helm/app -f values-aws.yaml
  
```

## Chapter 13. Admin UI · 시스템 관리

Case A (WBX 있음): admin-ui.enabled=false → WBX data.json 관리 페이지 사용

Case B (WBX 없음): admin-ui.enabled=true → 내장 Thymeleaf Admin Console

### 13-1. 접속 정보

```
# Admin Console URL
https://wbx.kr/spring/admin/login

# context-path 설정 (application.yml)
server:
  servlet:
    context-path: /spring # Caddy 프록시와 일치

# 모든 URL에 /spring prefix가 자동 적용됨:
# /spring/admin/login (Admin Console)
# /spring/api/auth/login (REST API)
# /spring/health (Health Check)
```

### 13-2. 구현된 관리 화면 (11개)

- 로그인 (/admin/login) — 이메일+비밀번호 폼 인증
- 대시보드 (/admin) — 활성 사용자, 전체 사용자, 로그인 횟수, 역할 수
- 사용자 목록 (/admin/users) — 전체 사용자 테이블, 상태/잠금 표시
- 사용자 상세 (/admin/users/{id}) — 잠금 해제, 비밀번호 초기화, 활성/비활성 전환, 역할 할당/해제
- 역할 목록 (/admin/roles) — 역할 코드/이름/설명/시스템 여부
- 역할 상세 (/admin/roles/{id}) — 역할 수정, 권한 추가/삭제
- 권한 매트릭스 (/admin/permissions) — 역할별 모듈-액션-범위 권한 테이블
- 로그인 이력 (/admin/login-history) — 최근 50건, 성공/실패/IP/인증방법
- 감사 로그 (/admin/audit-logs) — 최근 100건, 액션/리소스/IP/상세
- 시스템 설정 (/admin/config) — Key-Value 설정 관리, 추가/수정 폼
- 시스템 상태 (/admin/system-health) — JVM Heap, 스레드, OS, Java, Spring Boot 버전

### 13-3. Security 설정 (2개 FilterChain)

```
// Admin → 세션 기반 폼 로그인
@Order(1) adminFilterChain:
  /admin/** → formLogin → Thymeleaf 화면

// REST API → JWT Stateless
@Order(2) apiFilterChain:
  /api/** → JwtFilter → JSON 응답
```

## Chapter A. 설정 레퍼런스 (application.yml)

```
wbx:
  spring:
    api-prefix: /api/myapp      # API URL prefix
    api-version: v1            # 버전
  jwt:
    secret: ${JWT_SECRET}
    expiration: 28800          # Access Token (초)
    refresh-expiration: 604800 # Refresh Token (초)
  mfa:
    enabled: true
    force-for-external: true
    totp-issuer: "WBX Platform"
  password:
    min-length: 8
    max-failed-attempts: 5
    expiry-days: 90
    history-count: 3
  approval:
    enabled: true
  notification:
    sse-enabled: true
    websocket-enabled: false
    heartbeat-seconds: 30
  cors:
    allowed-origins: ${CORS_ORIGINS:http://localhost:5173,http://localhost:3000}
  file:
    storage-type: local        # azure-blob | aws-s3 | gcp-storage
    upload-dir: ./uploads
  admin-ui:
    enabled: false             # true=내장 Admin
    path: /admin
    allow-roles: [SA]
  compat:
    error-format: fastapi      # detail 키
    list-key: items
```

# Happy Coding!

아큐라시스템 | [accura@accurasoft.co.kr](mailto:accura@accurasoft.co.kr)