

春景智能 QMES SaaS 平台 — 系统架构设计说明书

文档版本: V1.0

编写日期: 2025-07-17

目录

- [1. 第一性原理设计](#)
- [2. 整体架构概览](#)
- [3. 技术选型与决策](#)
- [4. 四端架构设计](#)
- [5. 多租户架构](#)
- [6. 若依RBAC权限架构](#)
- [7. 模块架构与分层](#)
- [8. 数据架构](#)
- [9. 安全架构](#)
- [10. 部署架构](#)
- [11. 扩展性设计](#)

1. 第一性原理设计

架构核心命题

系统架构的每个决策回答一个核心问题: "这个架构选择能帮项目做什么决策?"

架构决策	核心命题	决策价值
SaaS多租户	"能否让100个工厂共用一套系统而不互相干扰?"	降低每个工厂的IT成本 (从数十万降为每月数千)
RBAC 5级权限	"能否让不同角色只看到该看的数据?"	保证数据安全, 同时让每个人高效工作
四端分离	"不同用户在不同设备上是否需要不同的交互方式?"	平台运营者、工厂管理员、车间操作工、移动端各有最优UI
Laravel+PostgreSQL	"技术栈能否支撑未来3年的业务增长?"	选择成熟生态, 保证团队招聘和社区支持

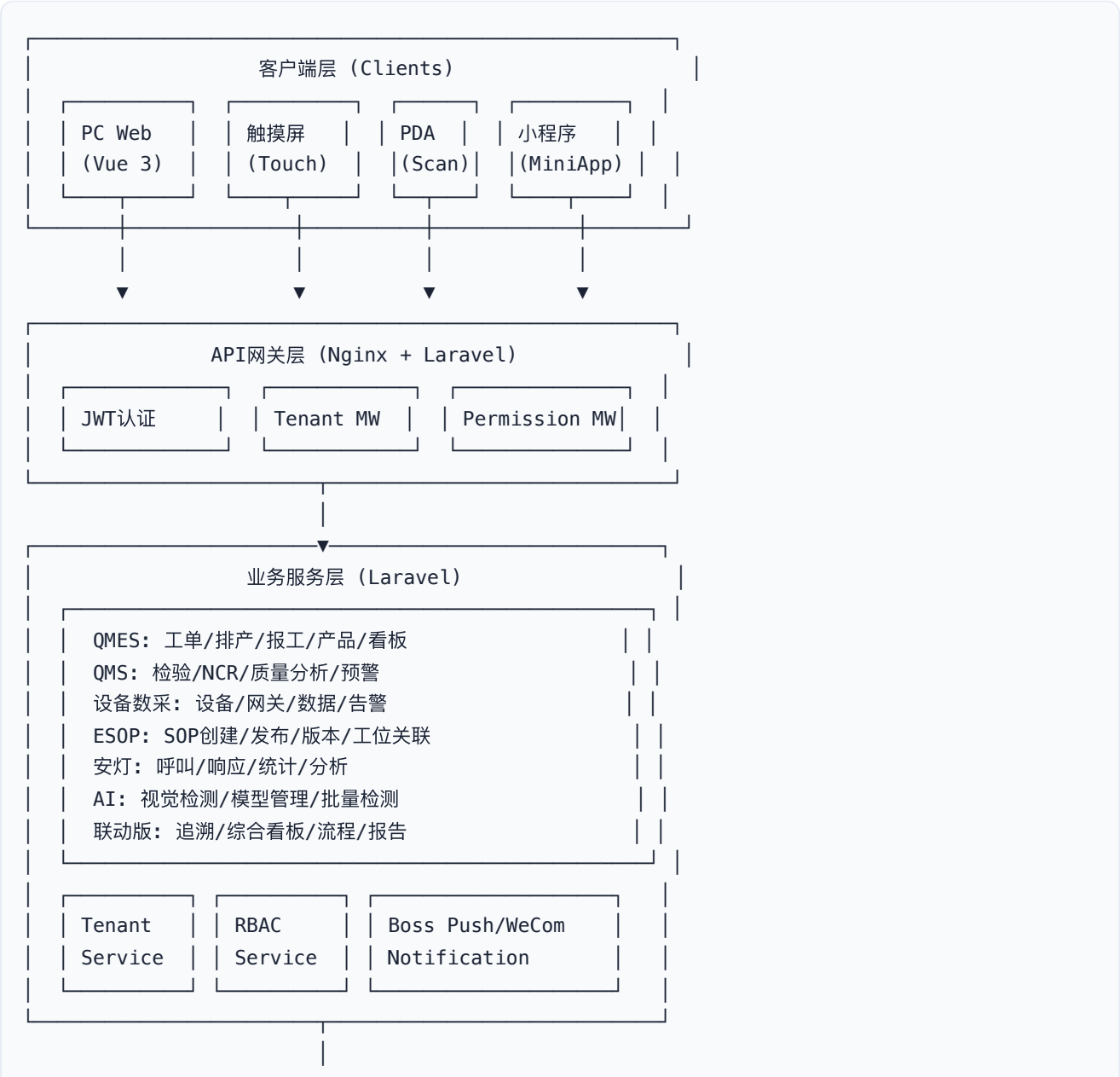
架构决策	核心命题	决策价值
Docker化部署	"部署一个新租户需要多久？"	从"半天"压缩到"5分钟"，运维效率提升50倍

架构原则

- 1. 隔离优先：租户间的数据和资源必须严格隔离
- 2. 权限前置：每个API请求必须在到达业务逻辑前完成权限校验
- 3. 可观测性：所有请求记录日志、监控耗时、追踪全链路
- 4. API驱动：所有功能通过RESTful API暴露，前端/移动端/第三方通过同一套API

2. 整体架构概览

2.1 分层架构图





2.2 请求生命周期



3. 技术选型与决策

3.1 后端技术栈

技术	版本	选型理由
PHP	8.2+	Laravel最优生态、协程支持、团队技术积累
Laravel	11.x	成熟的ORM/中间件/队列/事件系统
PostgreSQL	16	JSONB支持、窗口函数、CTE、GIS扩展能力
Redis	7.2	缓存/队列/Session/限流
JWT (tymon/jwt-auth)	2.1	无状态认证、跨端共享Token
Sanctum	4.0	API Token管理、SPA认证
GuzzleHttp	7.0	HTTP客户端 (AI API调用/企微推送)
php-mqtt	2.0	MQTT协议支持 (设备数据采集通道)
dompdf (barryvdh)	3.0	PDF报告生成 (追溯报告)

3.2 前端技术栈

技术	版本	选型理由
Vue 3	3.x	Composition API、性能优越
Element Plus	2.x	企业级UI组件库、中后台成熟方案

技术	版本	选型理由
Vue Router	4.x	动态路由、路由守卫
Pinia	2.x	状态管理、用户/权限Store
Axios	1.x	HTTP请求、拦截器
ECharts	5.x	甘特图/趋势图/饼图

3.3 基础设施

组件	技术	用途
Web服务器	Nginx 1.25	反向代理、静态资源、HTTPS终止
容器编排	Docker Compose	开发/测试/生产统一部署
队列	Laravel Horizon + Redis	异步任务（AI回调/企微推送/告警检查）
定时任务	Laravel Schedule	网关心跳超时检测、告警升级、质量预警

4. 四端架构设计

4.1 平台管理端 (Admin Portal)

访问方式: `admin.qmes.cn`

用户: 春景平台运营团队

技术实现: Vue 3 + Element Plus + Admin Layout

路由前缀: `/api/admin/`

权限中间件: `permission:qmes:admin`

核心功能模块:

platform-admin/	
├─ 租户管理	→ TenantController
├─ 套餐管理	→ TenantController::plans()
├─ 运营概览	→ OperationController::summary()
├─ 租户趋势	→ OperationController::tenantTrend()
├─ 收入趋势	→ OperationController::revenueTrend()
├─ 系统健康	→ OperationController::systemHealth()
├─ 版本发布	→ ReleaseController
├─ 公告管理	→ NotificationController
└─ 平台监控	→ OperationController

4.2 租户管理端 (Tenant Admin Portal)

访问方式: `{tenant}.qmes.cn`

用户: 工厂IT/系统管理员

技术实现: 与使用端共用Vue SPA, 通过权限路由控制可见性

路由前缀: `/api/tenant/*`

核心功能模块:

```
tenant-admin/
├── 用户管理      → UserController (tenant.users)
├── 角色管理      → RoleController (tenant.roles)
├── 菜单管理      → MenuController (tenant.menus)
├── 部门管理      → DeptController (tenant.depts)
├── 看板          → DashboardController (tenant.dashboard)
├── 工单统计      → WorkOrderController::statistics()
└── AI API配置    → Tenant模型扩展字段
```

4.3 租户使用端 (Tenant App Portal)

访问方式: `{tenant}.qmes.cn/app/*`

用户: 车间主任/质检员/操作工/设备主管

技术实现: 同一Vue SPA, 按权限显示不同菜单

路由前缀: `/api/app/*` + 各业务模块API

核心功能模块: 7个产品的全部业务功能

4.4 移动端 (Mobile)

访问方式: `{tenant}.qmes.cn/touch/*` | `{tenant}.qmes.cn/pda/*`

用户: 操作工/班组长

技术实现: 独立Vue路由, 触摸屏自适应布局

路由:

```
/touch/report      → 触摸屏报工页面
/touch/report/detail → 报工详情
/touch/report/defect → 不良登记
/pda/scan           → PDA扫码页面
```

移动端特性:

- 触摸屏大按钮设计, 适合手套操作
- 自动检测触摸设备 + 小屏幕重定向
- 离线缓存关键数据

4.5 条码服务架构 (BarcodeService)

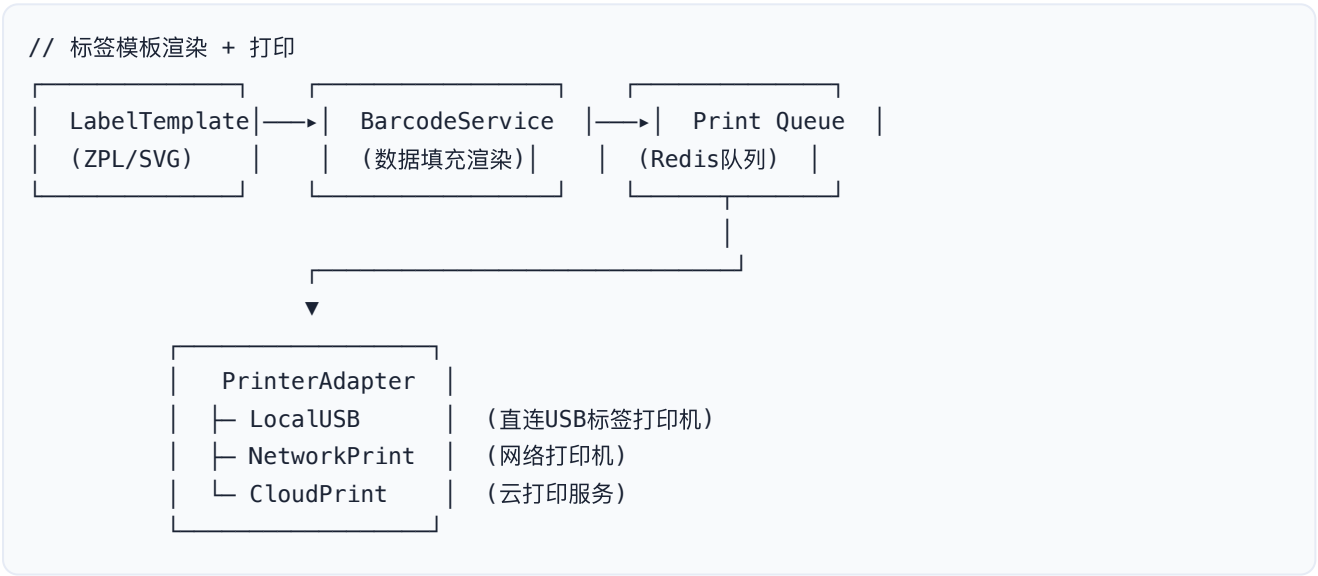
核心命题: "条码生成和打印如何与业务系统无缝集成?"

条码服务作为平台基础服务, 为工单、产品、物料、设备和仓储提供统一的条码生成、解析和打印能力。

4.5.1 条码生成链路

```
// BarcodeService 核心调用链路
客户端请求 → BarcodeController@generate
→ BarcodeService::generateBarcode($ruleId, $params)
  → 1. 加载 BarcodeRule (编码规则定义)
  → 2. 按规则组合[前缀+日期+流水号+产品代码]
  → 3. 调用 BarcodeGenerator (Code128/QR/DataMatrix)
  → 4. 生成条码图片/SVG/base64
  → 5. 持久化到 barcode_records 表
  → 6. 触发 BarcodeGenerated 事件
    → 监听器: PrintJobDispatcher (推送打印任务到队列)
    → 监听器: TraceLinkRecorder (记录追溯关联)
→ 返回条码数据(编码/图片/标签模板)
```

4.5.2 标签打印架构



4.5.3 条码解析服务

```
// 扫码枪/PDA → 条码解析 → 业务联动
PDA扫描条码 → BarcodeController@resolve
→ BarcodeService::resolve($barcodeContent)
  → 1. 匹配 BarcodeRule 判断编码格式
  → 2. 解析出 [前缀, 日期, 流水号, 产品代码]
  → 3. 查询关联业务记录 (WorkOrder/Product/Inventory)
  → 4. 返回业务对象 + 预定义操作URL
→ 前端根据返回内容自动跳转
```

条码格式规范:

条码用途	格式	示例
工单条码	WO + YYYYMMDD + 4位流水	WO202507170001
产品条码	PROD + 产品编码	PROD-A001-001

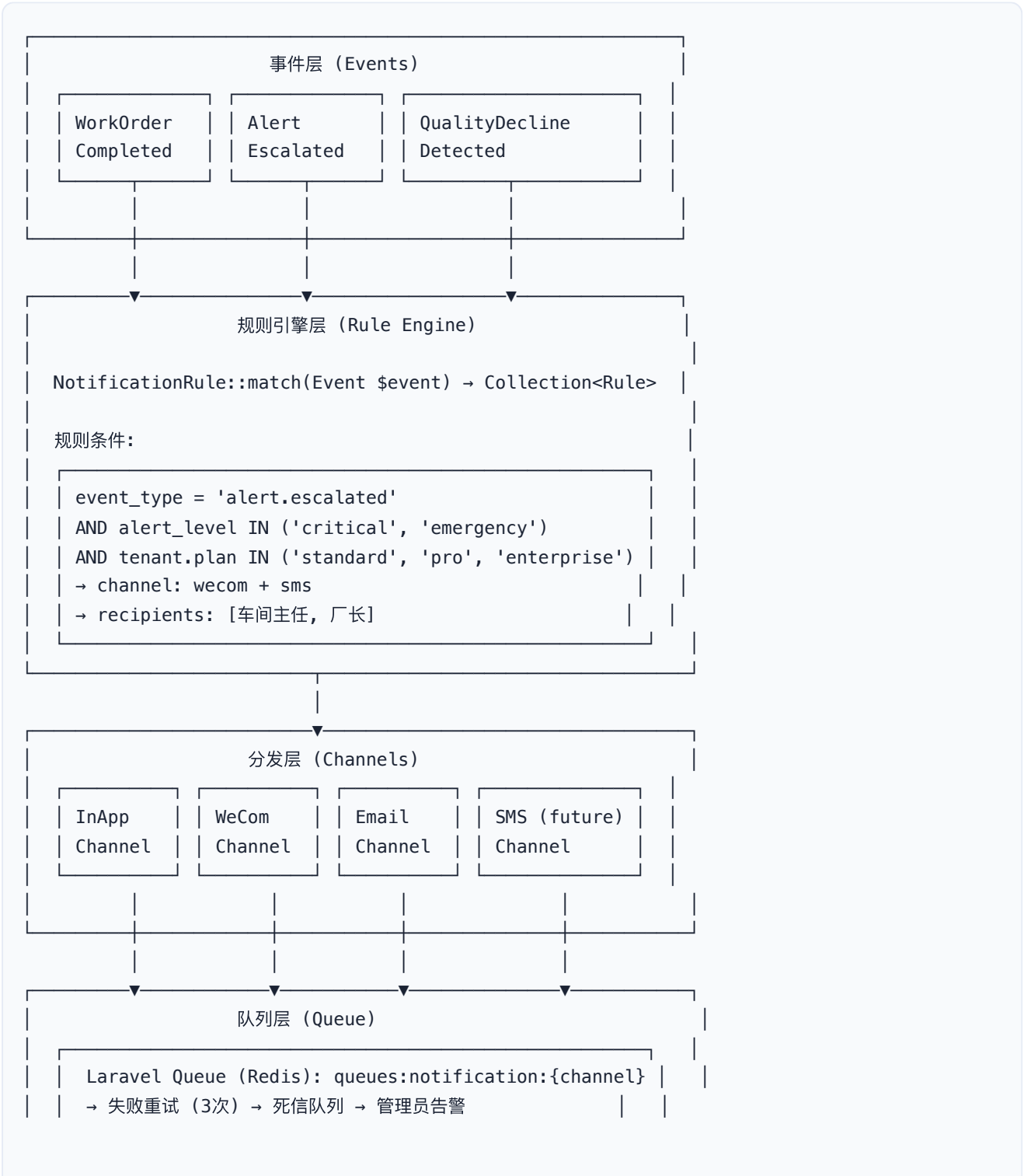
物料条码	MAT + 物料编码 + 批次	MAT-RAW-001-B202507
设备条码	DEV + 设备编码	DEV-CNC-001
托盘条码	PAL + 日期 + 流水	PAL202507170001

4.6 消息中心架构 (NotificationCenter)

核心命题: "如何确保业务事件可靠地触达正确的人?"

消息中心采用事件驱动架构, 将业务事件与通知渠道解耦, 通过规则引擎实现灵活的消息路由。

4.6.1 整体架构



4.6.2 通知模型定义

```
// 通知数据结构
Notification {
  id, tenant_id,
  title, body,
  channel: 'inapp' | 'wecom' | 'email' | 'sms',
  event_type: 'work_order.completed' | 'alert.escalated' | ...,
  recipients: [user_id, role_id, dept_id],
  template_id,           // 关联通知模板
  variables: {           // 模板变量填充
    'order_no': 'W0202507170001',
    'alert_level': 'critical'
  },
  status: 'pending' | 'sent' | 'failed',
  sent_at, read_at
}
```

4.6.3 渠道适配器模式

```
// 每个渠道实现同一个接口
interface NotificationChannel {
  public function send(Notification $notification): bool;
}

class WeComChannel implements NotificationChannel {
  public function send(Notification $notification): bool
  {
    // 调用企微Webhook API
    // 支持 text / markdown /图文消息类型
  }
}

class InAppChannel implements NotificationChannel {
  public function send(Notification $notification): bool
  {
    // 写入 user_notifications 表
    // 用户登录时在站内查看
  }
}
```

4.7 BI引擎架构 (BI & Dashboard Engine)

核心命题："如何让用户从业务数据中快速构建可视化看板？"

BI引擎采用数据源→数据集→图表→看板的四层架构，将数据查询与展示彻底解耦。

4.7.1 四层架构



4.7.2 缓存策略

```
// BI数据缓存 - 避免重复查询业务表
// 缓存键: bi:chart:{chart_id}:{filter_hash}
// TTL: 按chart配置 (默认5分钟)
// 失效策略: 定时过期 + 手动刷新
//
// 高频数据 (看板首页): 每30秒自动刷新
```

```
// 低频数据（月报分析）：手动触发刷新
// 实时监控数据：跳过缓存直接查询
```

4.7.3 数据查询隔离

BI引擎的数据查询走独立数据库连接（只读副本），确保复杂聚合查询不影响线上事务性能。ChartDataService 统一管理查询逻辑：

```
class ChartDataService
{
    public function getData(ChartConfiguration $chart, array $filters): array
    {
        $cacheKey = $this->buildCacheKey($chart, $filters);
        return Cache::remember($cacheKey, $chart->cache_ttl, function () use ($chart, $filters) {
            $dataset = $chart->dataset;
            // 使用只读连接执行查询
            return DB::connection('pgsql_read')
                ->select($dataset->compiled_sql, $filters);
        });
    }
}
```

4.8 行业模板架构 (Industry Template System)

核心命题：“如何通过配置驱动实现一套代码适配多行业的SaaS架构？”

行业模板架构在现有四端架构和业务模块之间增加一层“行业配置适配层”，该层不修改任何业务逻辑，而是通过JSON配置驱动前端渲染和后端功能开关。行业模板的加载在中间件层完成，早于任何业务控制器调用。

4.8.1 整体架构位置

```
请求入口 (Nginx)
↓
API网关层
├── JWT认证
├── TenantMiddleware (提取tenant_id)
├── IndustryTemplateMiddleware ← ★ 新增：行业模板加载
│   ├── 查询 tenant_industry_settings
│   ├── 加载 industry_templates.config
│   ├── 深度合并租户overrides
│   ├── 与License模块开关合并
│   └── 写入 app('industry_config')
├── PermissionMiddleware (权限校验)
↓
业务服务层 (各模块不变)
├── QMES/QMS/设备数采/ESOP/安灯/AI/联动版/...
└── 引用 industry_config 和 feature_flags
```

4.8.2 基座层加载流程

```
// IndustryTemplateMiddleware 核心逻辑
public function handle(Request $request, Closure $next): Response
{
    $tenant = tenant();
    if (!$tenant) {
        return $next($request); // 平台请求跳过
    }

    // 1. 查询租户行业设置（含模板预加载）
    $setting = TenantIndustrySetting::with('template')
        ->where('tenant_id', $tenant->id)
        ->first();

    if (!$setting) {
        // 2. 无设置时使用系统默认配置
        $config = config('industry.default');
    } else {
        // 3. 获取模板配置
        $templateConfig = $setting->template ? $setting->template->config : [];
        // 4. 深度合并：模板 + 租户覆盖
        $config = array_merge_recursive($templateConfig, $setting->overrides);
    }

    // 5. 写入容器，全局可访问
    app()->instance('industry_config', $config);

    // 6. License模块联动（License最终控制权）
    $licenseModules = $tenant->license->modules ?? [];
    $featureFlags = $config['feature_flags'] ?? [];
    $activeFlags = array_merge($featureFlags, $licenseModules);
    app()->instance('feature_flags', $activeFlags);

    return $next($request);
}
```

关键设计：

- 行业模板中间件位于 TenantMiddleware 之后、PermissionMiddleware 之前
- 配置使用 `array_merge_recursive` 实现深度合并
- License模块开关具有最高优先级，可覆盖模板配置
- 未设置模板的租户使用 `config('industry.default')` 兜底

4.8.3 JSON配置驱动架构

行业模板的核心是一个7维度的JSON配置结构，所有业务模块通过读取 `app('industry_config')` 来动态调整行为：

```
{
    "menu":          { "hidden_routes": [], "custom_order": {}, "labels": {} },
```

```
"dict_presets": { "work_center_type": [], "defect_categories": [] },
"terminology": { "work_order": "生产工单", "process": "机台" },
"dashboard": { "cards": [{ "title": "机台稼动率", "type": "gauge", "source": "equipment.i
"feature_flags": { "enable_mold_management": true },
"reports": { "daily_production": { "template_id": "report_injection_daily" } },
"workflow": { "patrol_inspection": { "enabled": true, "frequency": "每2小时" } }
}
```

4.8.4 界面文案实时替换架构

提供 `industry_terms()` 辅助函数和 Blade 指令 `@industry()`，在运行时将硬编码文案替换为行业专用术语：

```
function industry_terms($key, $default = null) {
    $config = app('industry_config');
    return $config['terminology'][$key] ?? $default ?? trans("terms.$key");
}
```

4.8.5 租户入驻初始化流程

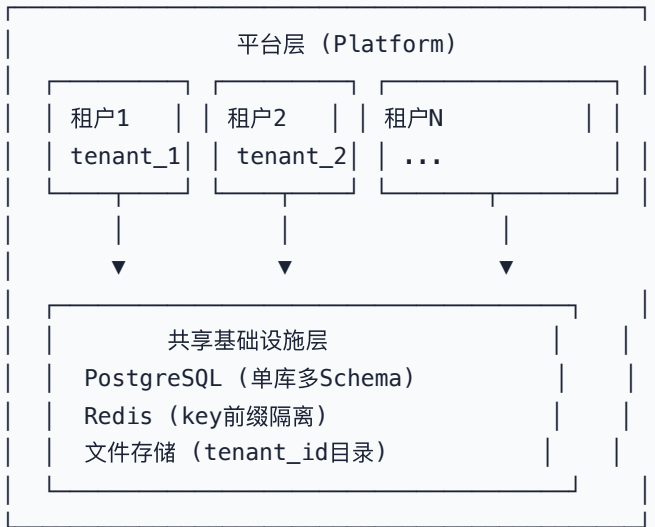
```
租户注册选择行业(industry_code)
↓
查询该行业默认模板 (is_default=1)
↓
创建 tenant_industry_settings 记录
↓
初始化预设字典数据 (标记来源=industry_template)
↓
在License中写入行业默认模块
↓
行业配置加载 → 租户即可使用行业定制化的界面
```

4.8.6 现有架构影响评估

维度	影响	说明
新增中间件	1个 (IndustryTemplateMiddleware)	性能可忽略，单次查询可缓存
新增表	3张	industries, industry_templates, tenant_industry_settings
新增辅助函数	1个 (industry_terms)	替代硬编码文案
业务模块改动	0	只需引用 industry_config 即可
前端改动	中等	菜单/术语/功能开关改为动态读取
缓存策略	新增 industry_config 缓存键	tenant_id 级别，模板变更时清除

5. 多租户架构

5.1 租户隔离模型



5.2 租户数据隔离策略

数据隔离级别: Row-Level + 全局Scope

```
// 1. 所有业务表包含 tenant_id 字段
Schema::create('work_orders', function (Blueprint $table) {
    $table->foreignId('tenant_id')->constrained();
    // ...
});

// 2. TenantScope 全局作用域自动追加 where tenant_id = ?
public function apply(Builder $builder, Model $model): void
{
    $tenantId = app('current_tenant_id');
    if ($tenantId) {
        $builder->where('tenant_id', $tenantId);
    }
}

// 3. 辅助方法简化查询
public function scopeByTenant($query, $tenantId)
{
    return $query->where('tenant_id', $tenantId);
}
```

5.3 租户套餐限制

```
// TenantMiddleware 检测租户状态
if ($tenant->status !== 'active') { /* 返回403 */ }
```

```
if ($tenant->isExpired()) { /* 返回403 */ }

// 创建用户时检测配额
public function canAddUser(): bool
{
    return $this->users()->count() < $this->max_users;
}
```

6. 若依RBAC权限架构

6.1 权限模型结构



6.2 五级数据权限

DataScope值	含义	实现方式
1	仅本人数据	<code>where('created_by', auth()->id())</code>
2	本部门数据	<code>whereIn('dept_id', 本部门所有用户ID)</code>
3	本部门及以下	<code>whereIn('dept_id', 本部门+子部门所有用户ID)</code>
4	全部数据	不追加where条件
5	自定义	通过自定义字段指定可见范围

6.3 权限标识体系

完整权限标识列表（核心权限约60+个）：

模块	权限标识
平台管理	qmes:admin, qmes:tenant:list/create/edit/delete/suspend
运营	qmes:operation:summary/release/notification
工单	qmes:work-order:list/create/edit/delete/start/report/complete
质检	qmes:inspection:list/create, qmes:quality:trend/dashboard
检验模板	qmes:inspection-template:list/create/edit/delete
NCR	qmes:ncr:list/create/edit

模块	权限标识
设备	qmes:device:list/create/edit/delete
设备数据	qmes:device-data:view/export
告警	qmes:alert:list/edit
网关	qmes:gateway:list/create/edit/delete
SOP	qmes:sop:list/create/edit
安灯	qmes:andon:list/create/edit
AI视觉	qmes:ai-vision:list/create/delete
AI模型	qmes:ai-model:list/create/edit/delete
追溯	qmes:trace:view/export
联动	qmes:integrated:dashboard
老板	qmes:boss:alert-config
产品	qmes:prod:product:list/create/edit/delete
工艺	qmes:prod:routing:list/create/edit
排程	qmes:prod:schedule:list/create/edit
缺陷码	qmes:prod:defect:list/create/edit/delete
自定义字段	qmes:custom-field:list/create/edit/delete

6.4 权限校验流程

请求 → PermissionMiddleware

→ 是否有 auth()->user()? → 否: 401

→ 是否超管? → 是: 放行

→ user.getAllPermissions() 获取所有权限

→ 遍历 \$permissions 参数, 匹配则放行

→ 无匹配: 403 "无权限访问"

7. 模块架构与分层

7.1 目录结构

app/

├ Http/

| └ Controllers/

```

├── Controller.php # 基础控制器
├── Api/
│   ├── AuthController.php # 认证
│   ├── TenantController.php # 租户管理
│   ├── TenantRegisterController.php # 租户注册
│   ├── UserController.php # 用户管理
│   ├── RoleController.php # 角色管理
│   ├── MenuController.php # 菜单管理
│   ├── DeptController.php # 部门管理
│   ├── DashboardController.php # 看板
│   ├── WorkOrderController.php # 工单
│   ├── ProductController.php # 产品
│   ├── ProductRoutingController.php # 工艺路线
│   ├── ProductionScheduleController.php # 排程
│   ├── InspectionController.php # 检验
│   ├── InspectionTemplateController.php # 检验模板
│   ├── NcrController.php # NCR
│   ├── DeviceController.php # 设备
│   ├── DeviceDataController.php # 设备数据
│   ├── DeviceAlertController.php # 设备告警
│   ├── GatewayController.php # 网关
│   ├── SopController.php # SOP
│   ├── AndonController.php # 安灯
│   ├── AiVisionController.php # AI视觉
│   ├── AiModelController.php # AI模型
│   ├── IntegratedController.php # 联动版
│   ├── OperationController.php # 运营
│   ├── ReleaseController.php # 发布
│   ├── NotificationController.php # 公告
│   ├── PaymentController.php # 支付
│   ├── CustomFieldController.php # 自定义字段
│   ├── DefectCodeController.php # 缺陷码
│   ├── BossAlertController.php # 老板告警
│   └── BossPushCheckController.php # 老板推送检查
├── Middleware/
│   ├── TenantMiddleware.php # 租户中间件
│   └── PermissionMiddleware.php # 权限中间件
├── Models/
│   ├── Tenant.php, User.php, Role.php, Menu.php, Dept.php
│   ├── WorkOrder.php, WorkReport.php
│   ├── Product.php, ProductRouting.php, ProductionSchedule.php
│   ├── InspectionRecord.php, InspectionTemplate.php, NcrRecord.php
│   ├── Device.php, DevicePoint.php, DeviceData.php, DeviceAlert.php, Gateway.php
│   ├── SopDocument.php, SopVersion.php, AndonCall.php
│   ├── AiVisionTask.php, AiModel.php
│   ├── DefectCode.php, CustomField.php, CustomFieldValue.php
│   ├── BossPushConfig.php, BossAlertLog.php, BossAlert.php
│   ├── PaymentOrder.php, OperLog.php, Release.php, Notification.php
│   ├── InventoryRecord.php, Inventory.php
│   └── Scopes/TenantScope.php
├── Services/
│   ├── TenantService.php # 租户创建/初始化/套餐管理
│   └── PermissionService.php # 权限/数据权限计算

```

└─ Helpers/
└─ helpers.php

全局辅助函数

7.2 模块耦合设计

模块	依赖关系	解耦方式
QMES	无外部依赖	核心引擎
QMS	依赖QMES工单	source_type + source_id 多态关联
设备数采	依赖QMES部门	部门ID做关联
ESOP	依赖QMES产品	product_id做关联
安灯	无外部依赖	工作站ID独立体系
AI	依赖租户配置	ai_api_url / ai_api_key 租户级配置
联动版	依赖QMES+QMS	通过工单号/批次号跨模块查询

8. 数据架构

8.1 数据库架构

- 数据库引擎: PostgreSQL 16
- 字符集: UTF-8 (utf8mb4)
- 主键策略: 自增整数ID (单表不超过1亿行)
- 索引策略: 所有外键建立索引, 高频查询字段复合索引

8.2 核心实体关系



8.3 缓存策略

数据	缓存键	TTL	更新策略
用户权限	perm:{user_id}	3600s	角色变更时清除

数据	缓存键	TTL	更新策略
菜单树	<code>menu_tree:{tenant_id}</code>	3600s	菜单变更时清除
配置项	<code>config:{tenant_id}:{key}</code>	600s	按需更新
看板数据	<code>dashboard:{tenant_id}</code>	60s	定期刷新

9. 安全架构

9.1 认证安全

机制	实现	配置
JWT认证	tymon/jwt-auth	TTL: 3600s, Refresh TTL: 20160s
API Token	Laravel Sanctum	用于第三方集成

9.2 数据安全

措施	说明
密码哈希	bcrypt (Hash::make)
SQL注入防护	Laravel Eloquent ORM 参数绑定
XSS防护	Vue 3 默认转义 + Content Security Policy
CSRF防护	JWT无状态 + CORS限定域名
请求速率限制	Laravel ThrottleMiddleware
敏感信息	不返回密码字段, JWT payload 不包含敏感信息

10. 部署架构

10.1 Docker Compose 部署

容器	镜像	端口	用途
nginx	nginx:1.25-alpine	8080:80	反向代理+静态资源
php-fpm	自定义 (php.dockerfile)	9000	Laravel应用
postgres	postgres:16-alpine	5432	关系数据库
redis	redis:7.2-alpine	6379	缓存+队列

容器	镜像	端口	用途
horizon	自定义 (php.dockerfile)	-	Laravel Horizon队列

10.2 部署流程

```
1. git clone / 拉取代码
2. 配置 .env (DB/REDIS/JWT等)
3. docker-compose up -d
4. docker-compose exec php-fpm composer install
5. docker-compose exec php-fpm php artisan migrate
6. docker-compose exec php-fpm php artisan db:seed
7. docker-compose exec php-fpm php artisan jwt:secret
8. 配置Nginx站点指向 frontend/dist/
```

11. 扩展性设计

11.1 水平扩展

组件	扩展方式
PHP-FPM	增加容器实例，Nginx upstream轮询
Redis	Redis Cluster 或 Redis Sentinel
PostgreSQL	PgBouncer连接池 + 读写分离
队列	Horizon支持多worker进程，按队列分组

11.2 功能扩展

扩展点	设计方式
自定义字段	JSON fields + CustomField 模型
插件系统	通过事件/监听器机制扩展
Webhook	企微推送采用可配置的webhook URL
AI模型	可插拔的AI API配置，支持不同服务商
第三方集成	所有功能均通过RESTful API暴露