# 🏗️ 海外电商 ERP 系统重构与架构升级方案

**版本**: 1.0

**状态**: 规划中

**目标**: 提升系统稳定性、可维护性及扩展性，解决开发环境痛点，为业务增长做技术储备。

## 1. 现状诊断 (Current State Analysis)

当前项目采用 **Vue 3 (Frontend)** + **Express (Backend)** + **Prisma (ORM)** 的全栈单体架构 (Monorepo)。

### ✅ 优势 (Strengths)

* **技术栈成熟**: 选用了 Node.js 生态中开发效率极高的 "黄金三角" (Express + Prisma + Zod)。
* **领域模型清晰**: schema.prisma 对跨国业务（多国家、多币种、复杂物流状态）的建模非常细致且专业。
* **前端现代化**: 使用了 Vue 3 Composition API 和 Tailwind CSS，符合现代前端标准。
* **数据完整性**: 后端普遍使用了 Zod 进行运行时参数校验，保证了入库数据的质量。

### ⚠️ 痛点与风险 (Weaknesses & Risks)

1. **开发环境脆弱**: 前端通过硬编码逻辑处理 API URL (src/api.js)，未利用 Vite Proxy，导致跨域配置复杂且与生产环境不一致。
2. **后端路由臃肿 (Fat Routes)**: backend/routes/ 承担了过多职责（校验、权限、业务逻辑、DB操作），导致代码难以复用和测试。
3. **缺乏类型安全**: 面对复杂的销售和物流数据结构，纯 JavaScript 开发容易导致字段拼写错误或类型不匹配，重构风险大。
4. **同步阻塞风险**: Excel 导入解析在主线程同步执行，大文件上传可能导致服务器假死。
5. **UI 组件复用率低**: 大量手写的 Table 和 Modal 组件，包含重复的 Tailwind 类名，维护成本高。

## 2. 重构路线图 (Refactoring Roadmap)

### 🟢 第一阶段：基础设施修复 (P0 - 立即执行)

**目标**: 统一配置管理，解决开发环境跨域痛点，确保环境一致性。

1. **配置 Vite Proxy (前端)**
   * **动作**: 修改 frontend/vite.config.js，配置 server.proxy 将 /api 请求转发至后端端口 (3000)。
   * **动作**: 简化 frontend/src/api.js，移除 URL 嗅探逻辑，baseURL 统一设为 /api。
2. **统一后端配置源**
   * **动作**: 强制所有环境变量（DB URL, JWT Secret, API Keys）仅通过 backend/config.js 导出。
   * **动作**: 移除 index.js 中手动加载 .env 的逻辑，统一由 config.js 处理并进行 Zod 校验。

### 🔵 第二阶段：后端架构分层 (P1 - 核心重构)

**目标**: 解耦业务逻辑与路由控制，引入 Service 层模式。

1. **引入 Service/Controller 层**
   * **重构目标结构**:  
     backend/  
     ├── controllers/ # 接收请求，解析参数，调用 Service，返回响应  
     ├── services/ # 纯业务逻辑（汇率计算、数据清洗、复杂查询）  
     ├── routes/ # 路由定义与中间件挂载  
     └── app.js
   * **实施**: 将 routes/salesData.js 中的逻辑拆分。例如，权限检查逻辑抽离为 Service 方法，汇率转换逻辑独立封装。

### 🟡 第三阶段：稳定性与类型安全 (P2 - 进阶)

**目标**: 消除运行时错误隐患，解决性能瓶颈。

1. **迁移至 TypeScript**
   * **动作**: 引入 typescript 和 ts-node。
   * **收益**: 利用 Prisma 生成的类型定义 (Prisma.SalesDataCreateInput)，在编译期发现字段错误。
2. **引入异步任务队列**
   * **集成**: Redis + **BullMQ**。
   * **场景**: 将 Excel 导入（salesImport.js）改为异步任务。用户上传后立即返回，后台 Worker 逐行解析入库，前端轮询进度。
3. **优化财务数据存储**
   * **策略**: 写入时即计算并存储“本位币金额” (cnyAmount) 和“当日汇率”。查询统计时直接聚合，不再实时换算。

### ⚪ 第四阶段：前端体验升级 (P3 - 优化)

**目标**: 提升代码复用率，统一 UI 风格。

1. **集成 UI 组件库**
   * **推荐**: **Shadcn-vue** (基于 Tailwind 的 Headless 组件) 或 Element Plus。
   * **动作**: 替换手写的 Modal 和 Form 组件，提升交互一致性。
2. **封装 API 层**
   * **动作**: 在 frontend/src/services/ 下按模块封装 API 请求（如 authService.js, salesService.js），组件内不再直接调用 axios.get。

## 3. 技术栈推荐 (Tech Stack Radar)

### ✨ 建议引入

| **类别** | **工具/库** | **用途** |
| --- | --- | --- |
| **语言** | **TypeScript** | 提供全栈类型安全，极大降低维护成本。 |
| **任务队列** | **BullMQ** | 处理耗时的 Excel 导入、邮件发送等后台任务。 |
| **UI 组件** | **Shadcn-vue** | 高度可定制的现代化组件库，配合 Tailwind 使用体验极佳。 |
| **数据表格** | **TanStack Table** | 处理复杂的排序、筛选、分页逻辑 (Headless UI)。 |
| **图表** | **ECharts** | 处理复杂的电商数据可视化（比 Chart.js 更强大）。 |
| **日志** | **Winston / Pino** | 结构化日志记录，便于后续接入日志分析系统。 |

### 🗑️ 建议移除/替换

* **Axios 裸调用**: 替换为 Service 层封装调用。
* **手动 CORS 逻辑**: 替换为 Vite Proxy + Nginx。
* **console.log**: 生产环境替换为专业的日志库。

## 4. 未来展望 (Future Considerations)

1. **多租户支持 (Multi-tenancy)**
   * 若业务扩展至多个独立事业部，需在数据库层引入 OrganizationID，实现数据逻辑隔离。
2. **审计日志 (Audit Logging)**
   * 针对财务 (Finance) 和物流 (Logistics) 的关键操作，建立 AuditLog 表，记录“谁(Who) 在何时(When) 修改了什么(What)”。
3. **CI/CD 自动化**
   * 建立自动化流水线：代码提交 -> 运行 Vitest 测试 -> 构建 Docker 镜像 -> 自动部署。
4. **AI 智能助手**
   * 利用现有的周报和销售数据，集成 LLM API，自动生成周报摘要或销售异常分析报告。