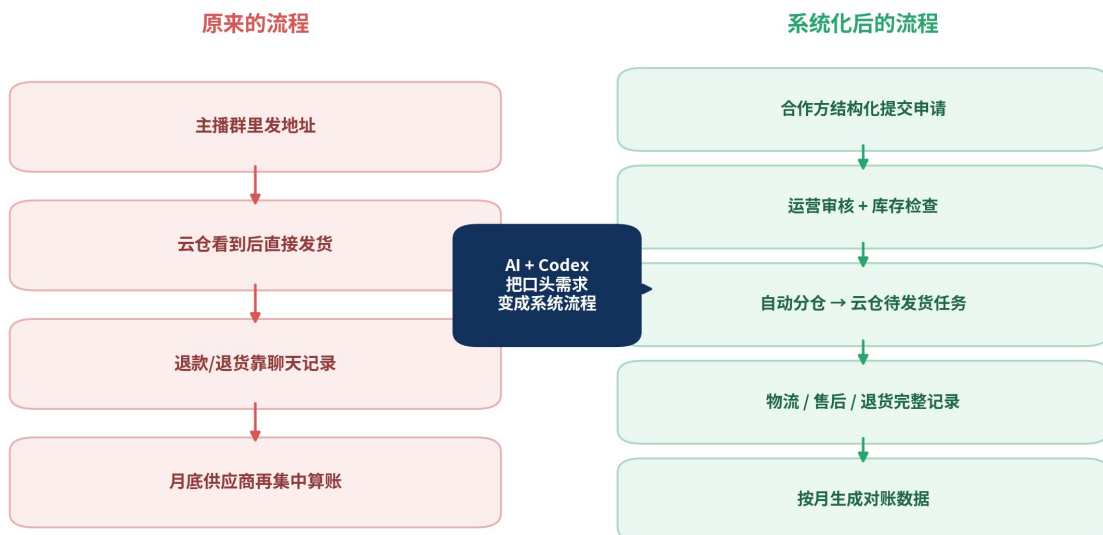


AI + Codex 真实业务落地案例

从微信群发货，到可追踪、可对账的供应链履约系统

美兰美捷美肤工作台 · v0.2.1

从“微信群 + 人工记忆”到“结构化业务闭环”



我的角色：需求收集 / 业务抽象 / AI 共创 / Agent 编排 / 验收与生产落地

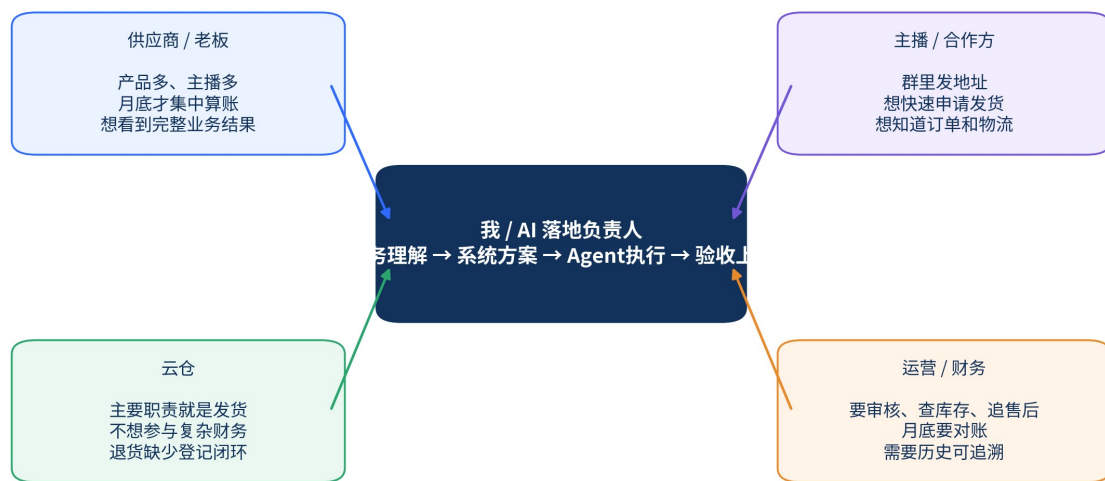
01 这不是一个“想做后台”的项目，而是一个真实业务问题

老板本身是供应商，合作商品多、主播多。平时发货的核心流程并不是系统，而是微信群：主播把收货地址、商品和数量发在群里，云仓也在群里，看见以后直接发货。供应商平时很少介入中间过程，通常到月底才统一核对。

真正的问题在月底集中爆发：发了什么、给谁发的、退了什么、哪些衣服退回云仓、售后是否重新入库，都要重新翻聊天记录、对 Excel、问云仓。云仓的职责又主要是“发货”，退货和售后没有形成完整登记闭环。

真实业务不是一个人的需求，而是多方诉求的交集

我的工作：收集、澄清、抽象、取舍，再把共识变成可落地的系统规则



我面对的不是一个需求方，而是供应商、主播、云仓、运营/财务四类不同诉求。

我先做的不是写代码，而是还原业务现场

- 谁在发起发货？谁确认？谁实际发货？
- 哪些信息现在只存在微信群？哪些信息月底才补？
- 退货、补发、换货到底由谁登记？
- 云仓真正愿意操作到什么程度？
- 供应商月底最需要看到的结果是什么？

真实业务问题 → AI 共创方案 → Codex 执行 → 验收上线

02 ：把“人话需求”翻译成可落地的系统规则

这个项目里，我不是拿到完整 PRD 再开发。我更像业务与技术之间的“翻译器”：先听各方怎么做事，再抽象规则、发现冲突、决定边界，最后把它变成 AI 和 Coding Agent 可以执行的任务。

业务人员原话	我进一步拆解	最后形成的系统能力
“月底算账太乱了”	需要把申请、发货、售后、退款按合作方和月份串起来	月度对账 + 明细导出 + 锁账快照
“地址都发群里”	缺少统一入口和结构化订单	外部申请端 + 订单状态 + 历史查询
“退回来的衣服没人登记”	退货必须关联原订单，并影响后续库存和对账	RETURN / EXCHANGE / RESHIPMENT 售后模型
“云仓只管发货”	不能让仓库承担复杂财务和后台操作	独立仓库工作区 + XLSX 批量履约
“产品和主播太多”	需要明确订单归属、SKU、仓库和库存关系	Product / SKU / Warehouse / Inventory 模型

我的核心工作不是“把所有需求都做进去”，而是判断哪些应该进系统、哪些应该保留原有习惯、哪些必须通过权限和数据规则隔离。

03 我怎么用 AI + Codex : 我负责判断, AI 负责加速

我如何用 AI + Codex 推进真实项目



我不会直接对 *Codex* 说“帮我做一个订单系统”。每个阶段都会先把问题变成明确任务，再给 *Agent* 指定代码检查范围、允许修改的模块、禁止事项、数据库要求、测试要求和验收标准。

ChatGPT / AI 主要帮我做什么

- 把模糊业务问题继续追问、补全边界和异常场景。
- 从完整数据流考虑：页面 → *API* → *Service* → *ORM/SQL* → *Database*。
- 比较多个方案，避免为了一个页面需求破坏整体架构。
- 把业务目标整理成 *Codex/Kimi Code* 可直接执行的工程 *Prompt*。

Codex / Kimi Code 主要帮我做什么

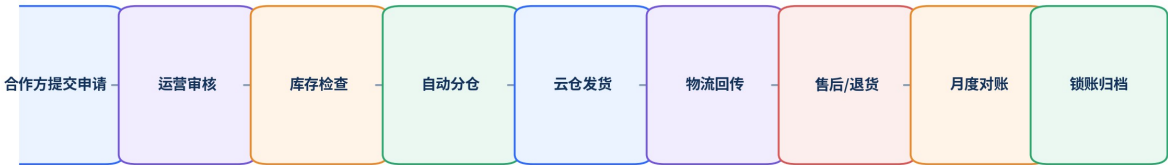
- 读取现有代码和数据库结构，不从零重复造系统。
- 按限定范围实现页面、*API*、*Migration* 和测试。
- 运行 *Build*、*Playwright*、*SQL* 验证和日志检查。
- 生产故障时根据 *Network*、*systemd journal*、*PostgreSQL error* 收集证据。

真实业务问题 → AI 共创方案 → *Codex* 执行 → 验收上线

04 最终落地的系统是什么

美兰美捷美肤工作台是一套围绕“合作方申请发货 → 内部审核 → 库存与仓库分配 → 云仓发货 → 物流 → 售后/退货 → 月底对账”的轻量履约系统。

落地后的完整业务闭环



所有关键节点都有状态、角色和数据记录，不再依赖微信群“谁记得就问谁”



三类使用者，各自只做自己最需要的事

角色	主要操作	不需要承担的事情
合作方/主播	提交发货申请、看自己的订单/物流/售后	不看库存、仓库内部数据和其他合作方
运营/管理员	审核、库存、商品、售后、对账、审计	不替云仓做实际拣货发货
云仓	看待发货任务、拣货、发货、录入/上传物流	不看成本、利润、月结敏感信息

05 我不是为了“功能多”，而是针对痛点做取舍

真实痛点 → 系统能力 → 业务便利

业务痛点	系统能力	带来的便利
群里发地址，容易漏单/翻记录	外部申请端 + 结构化订单	申请有记录、状态可查
产品/主播多，供应商中途看不清	订单归属 + 审核 + 物流链路	随时知道谁申请、发了什么
云仓只发货，不登记退货	售后 RETURN / EXCHANGE / RESHIPMENT	退货/补发不再脱离原订单
月底对账靠人工拼 Excel	按合作方 + 月份生成对账单	月底直接查历史和导出
仓库习惯 Excel，不想复杂操作	固定 XLSX 下载/上传 + 严格 ID 校验	保留仓库习惯，同时系统可追踪
AI 改代码容易越界	Prompt Guardrail + Build/Test/Verify	AI 加速但不失控

一个典型取舍：为什么仓库仍然保留 Excel

如果只从技术视角出发，很容易要求云仓“所有事情都进网页操作”。但真实业务里，仓库已经习惯批量用 *Excel*。我没有强行改变习惯，而是做成固定 *XLSX* 下载/上传流程，并通过 *batch_id*、*warehouse_id*、*fulfillment_line_id* 做严格校验。

这个设计体现的是落地思维：不是为了系统化而系统化，而是在业务可接受的操作方式上增加可追踪性和数据一致性。

另一个取舍：月底对账为什么需要“锁账快照”

如果月底导出的账单一直读取实时订单，历史订单被修改后，之前确认过的 *Excel* 就可能和系统不一致。因此我和 AI 一起把“月底对账很乱”继续抽象成 *Draft* → *Lock* → *Snapshot* → *Unlock/Relock* 的版本化流程。

06 这个系统给各方带来了什么便利

对象	以前	现在
供应商	月底才翻群、问仓库、拼 <i>Excel</i>	随时查订单、商品、物流、售后；月底按合作方直接对账
主播/合作方	每次在群里重复发地址和商品信息	通过独立入口申请，并查看自己的进度和物流
云仓	在聊天记录里找地址，只记录发出去了什么	直接处理待发货任务；批量 <i>XLSX</i> 保留原有操作习惯
运营/财务	靠人工记忆串联发货、售后、退款	审核、售后、对账和历史记录都在统一数据链中

最重要的变化不是“多了一个后台”，而是业务第一次有了统一事实来源：谁申请、谁审核、发了什么、从哪个仓发、有没有退货、月底怎么算，都能够沿着同一条数据链追溯。

系统价值

- 减少重复沟通和月底找记录的时间。
- 让供应商从“月底才知道结果”变成“中间过程也可见”。
- 让售后/退货不再脱离原订单。
- 让云仓继续用熟悉的操作方式，同时留下系统记录。
- 让业务逐渐从个人记忆变成标准流程。