

**系统设计报告**

**专业：软件工程**

**班级：实训**

**小组：第3小组**

**成员：20301002 党田乐**

**20301007 何军庭**

**20301024 王 圳**

**20301155 邹佳琪**

**20301164 柳萱莹**

**2023.06**

**目录**

[一、 系统技术架构 1](#_Toc2067165248)

[二、 针对新需求的改进 1](#_Toc855881970)

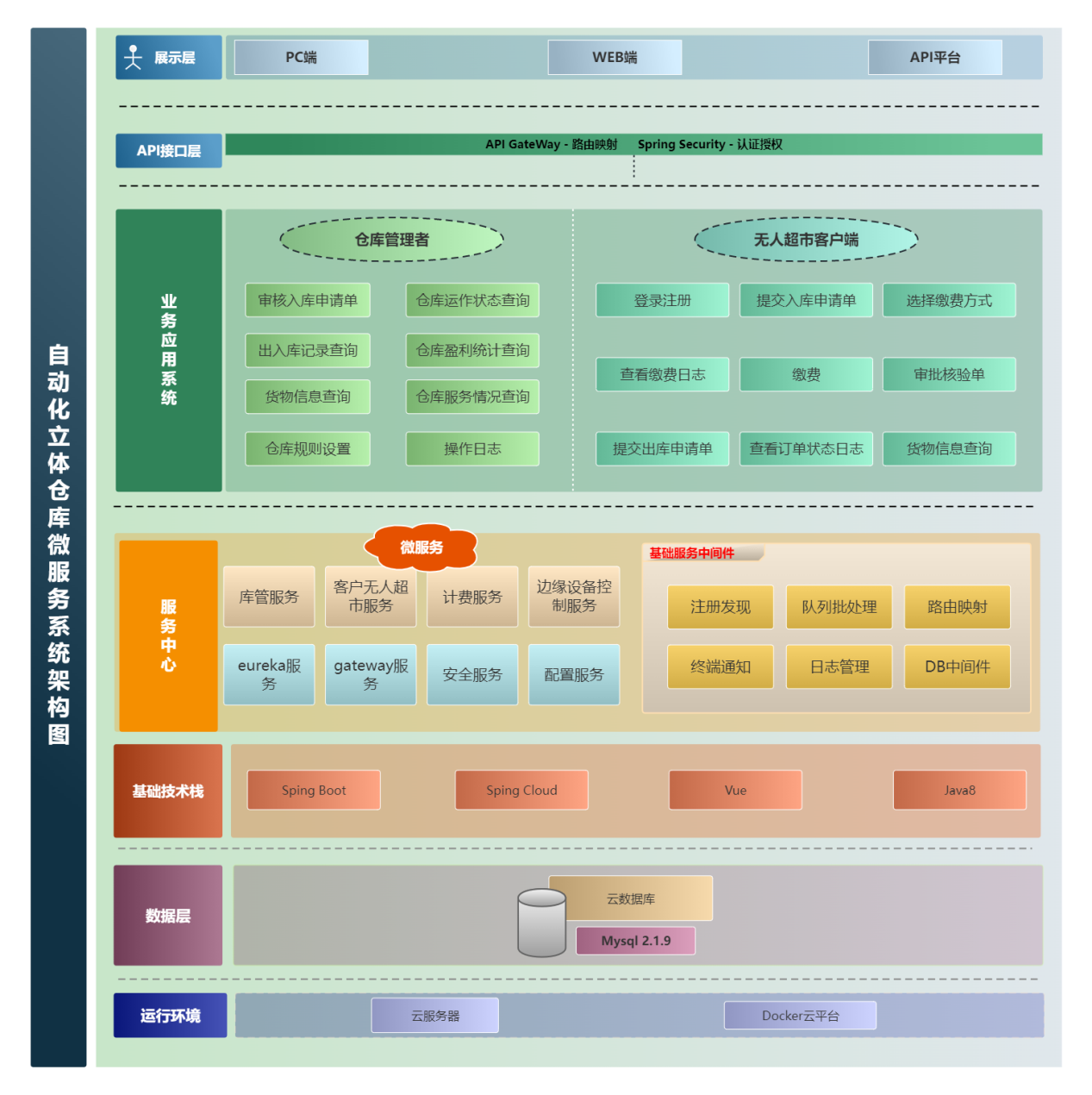
[三、 开发内容 7](#_Toc962802184)

[四、 架构设计的总结与思考 8](#_Toc527026343)

[五、 团队开发的总结与思考 8](#_Toc1509186573)

[六、 代码管理的总结与思考 9](#_Toc969377694)

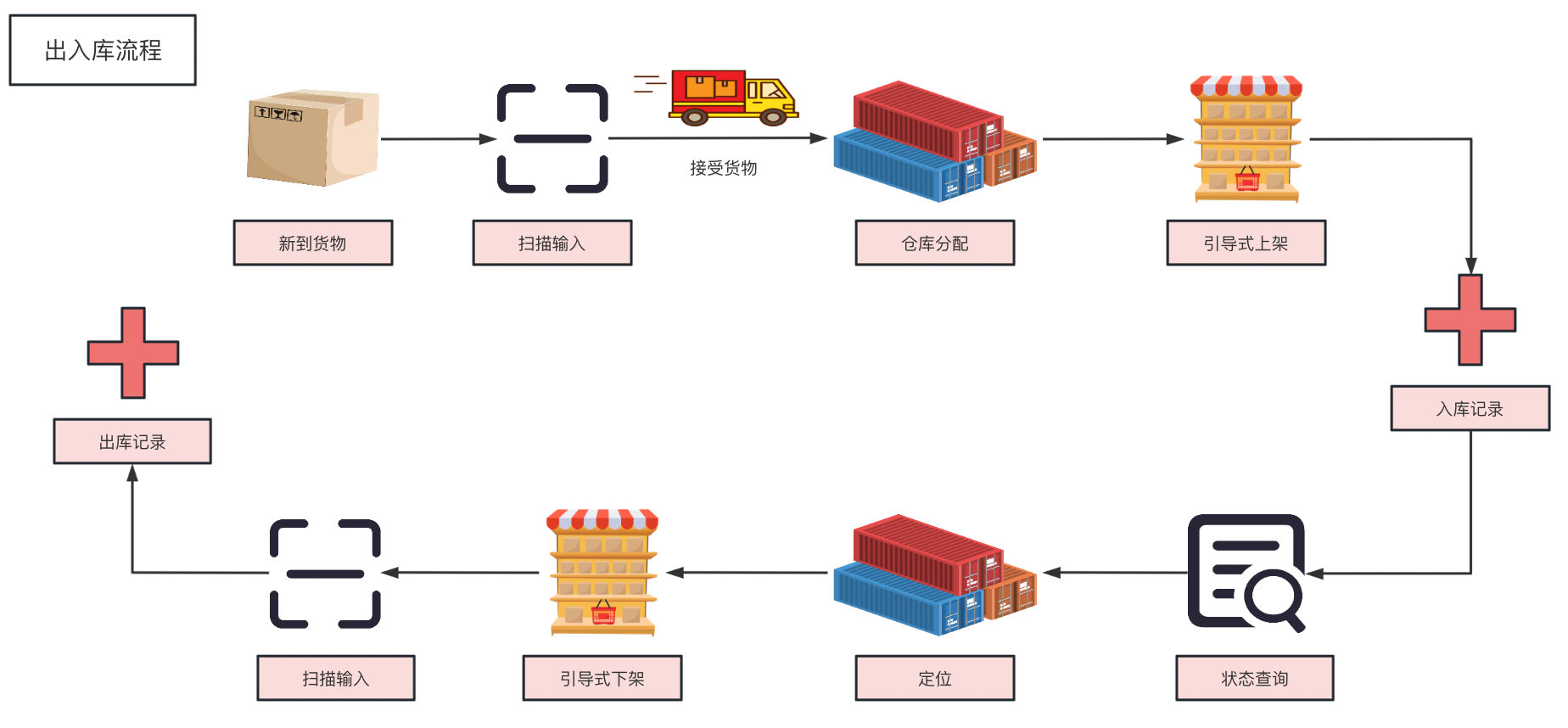
# 系统技术架构



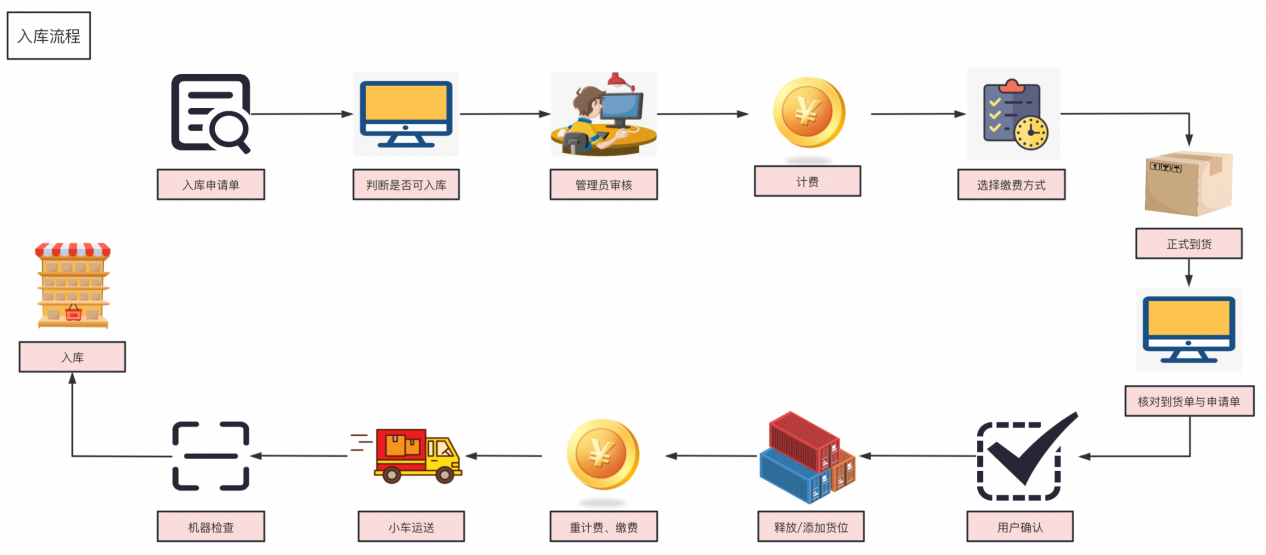
# 针对新需求的改进

### 2.1.1针对入库：

（1）旧需求下的入库流程



（2）新需求下的入库流程



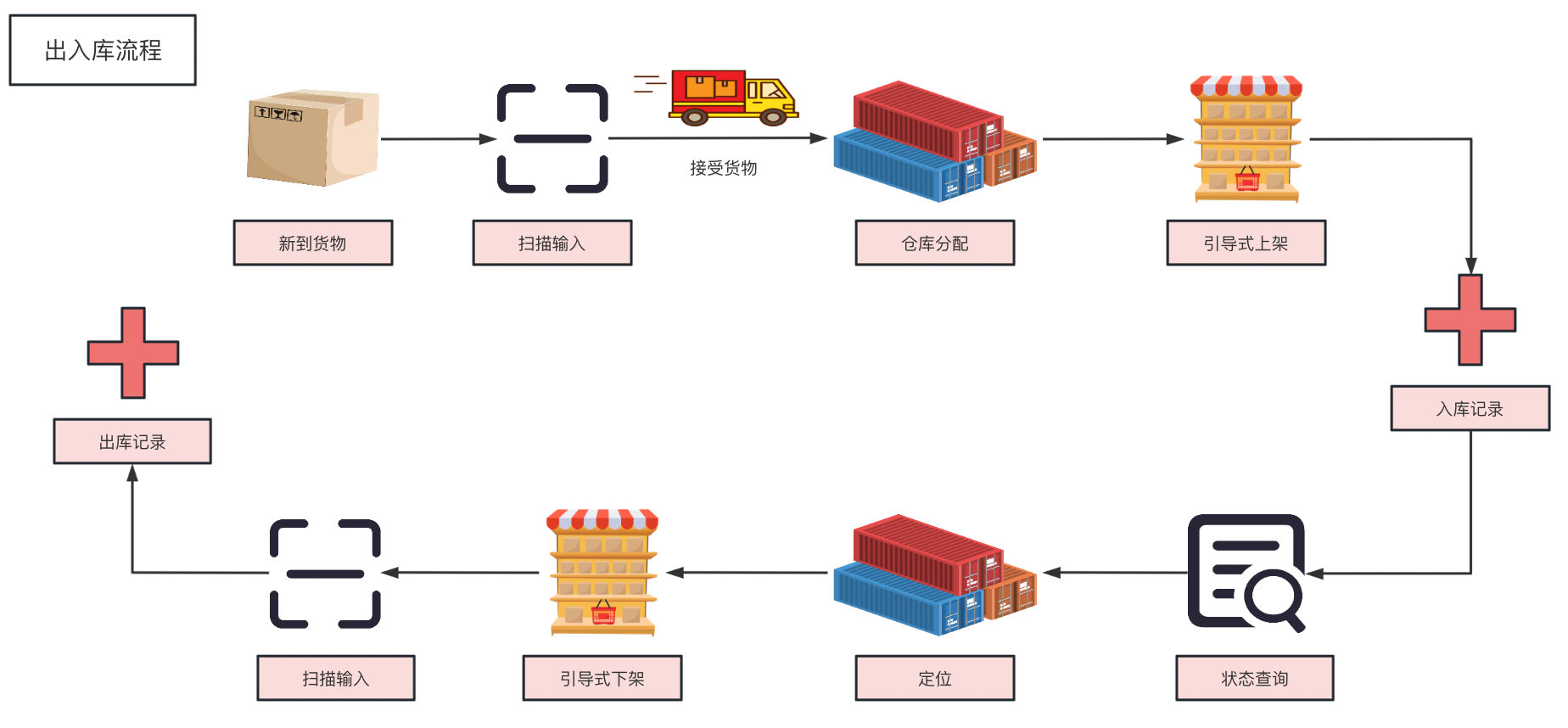
1. 数据库和代码的改变：

针对新的入库需求功能实现，主要区别在于order表状态的增多，以及InputService类中的checkIn函数（用于检查到货单与入库申请单），无需添加新的实体类、controller、service类等，而判断是否可入库、选择可用仓库柜、计费方式等均不变。因此以下列出了新入库需求功能实现的数据库和代码的改变。

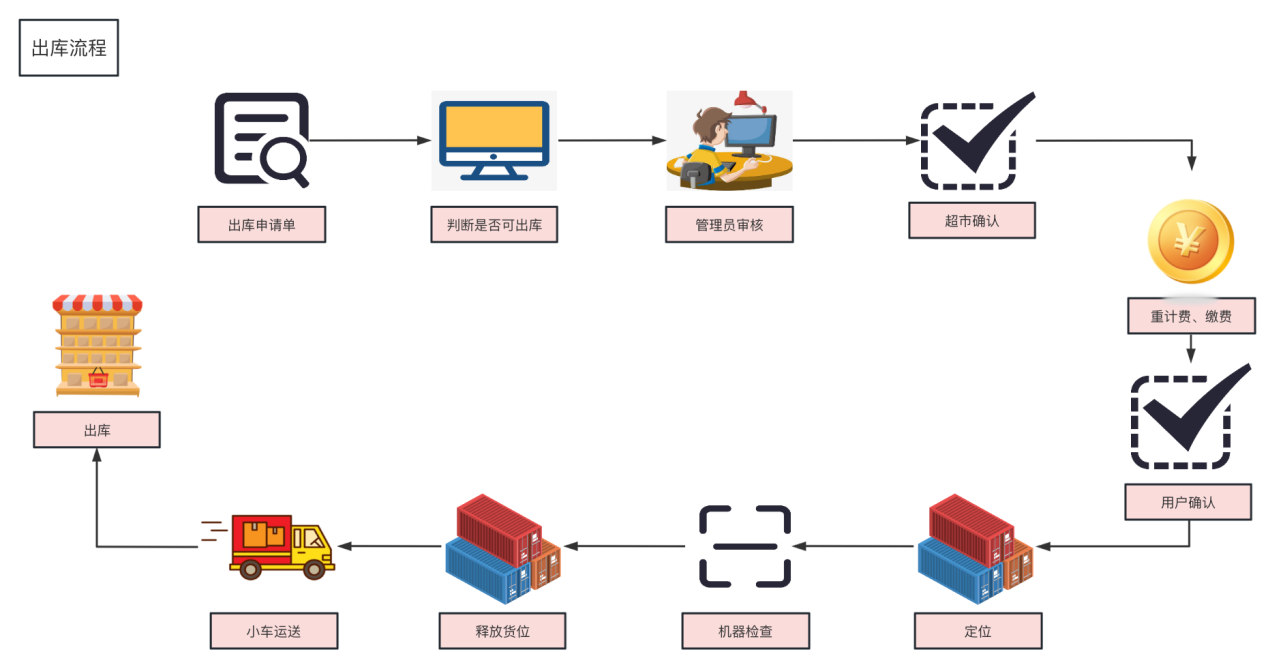
* 发来入库申请单 【数据库order表插入一条记录，状态为：系统审核状态】
* InputService判断可入库【修改order表的状态为：人工审核状态/未通过系统审核状态】
* 人工审核后【修改order表状态：待选择缴费方式/人工审核不通过】
* 超市选择缴费方式后 【修改order表状态为：已预留库位状态(即已选择缴费方式）】
* 到货单
* InputService核对到货单与申请单后 【修改order表状态为：入库清单待确认状态】
* 超市确认入库【修改order表状态为：核验单已确认状态】
* 释放/添加货位
* 重计费 【修改order表状态为：入库缴费状态】
* 缴费/扣费
* 小车运送【修改order表状态为：入库待确认状态】
* 机器检查【修改order表状态为：入库已确认状态】
* 入库

### **2.1.2 针对出库：**

（1）旧需求下的出库流程



（2）新需求下的出库流程



（3）数据库和代码的改变：

针对新的出库需求功能实现，主要区别在于save、order表状态的增多，无需添加新的实体类、controller、service类等，而判断是否可出库、重计费方式均不变。因此以下列出了新入库需求功能实现的数据库和代码的改变。

* 发来出库申请单 【状态为：出库系统审核状态】
* OutputService判断可出库【修改save表的状态为：人工审核状态/未通过系统审核状态】
* 人工审核后【修改save表状态：超市待确认/人工审核不通过】
* 超市确认出库【修改save表状态：待缴费】
* 重计费、超市缴费 【修改save表状态为：出库已确认状态】
* 定位
* 机器检查【修改save表状态为：等待出库】
* 小车运送【修改save表状态为：出库中】
* 出库【修改order表状态为：部分出库/全部出库】

## 2.2非功能需求的对比

### **2.2.1 spring security**

JwtUtil有createToken和verifyToken两个功能，分别用于在登录时创建token和其它接口时验证token。

其中createToken方法用于在用户成功登录后生成一个token，包含用户的ID和部分用户信息。

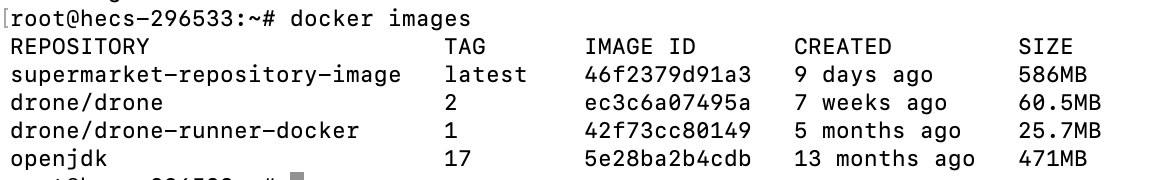
而verifyToken方法用于在接口调用时校验token是否有效和未过期。

JwtFilter是一个过滤器，它的主要作用是对请求进行拦截，并根据请求类型选择处理。在登录接口的请求中，JwtFilter会放行请求，而对其它请求则会检验token是否存在和是否已过期，如果token失效，JwtFilter会拦截请求，并返回相关信息，要求用户重新登录。这样的设计可以有效地保护系统安全，确保只有已登录用户才能访问系统接口，同时也避免了频繁向数据库请求验证用户信息，提高了系统响应速度。

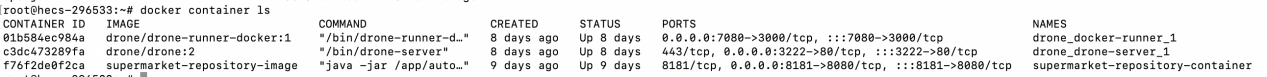
### **2.2.2 Docker**

为实现自动且快速的扩容和缩容，docker 容器可以在任意平台运行，不管是物理机还是虚拟机，不管是公有云还是私有云，甚至是个人电脑，所以我们的自动仓储系统容易做迁移和扩展。

打包项目jar文件到/target目录，编写Dockerfile，在云服务器上生成supermarket-repository-image镜像



启动容器：

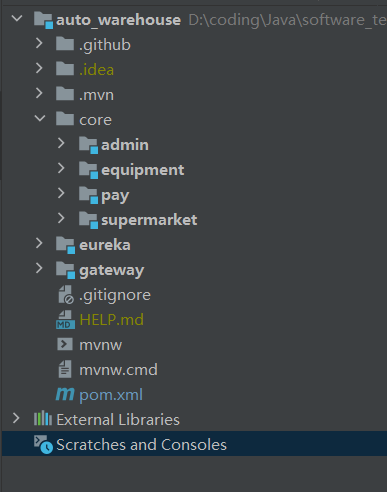


### 

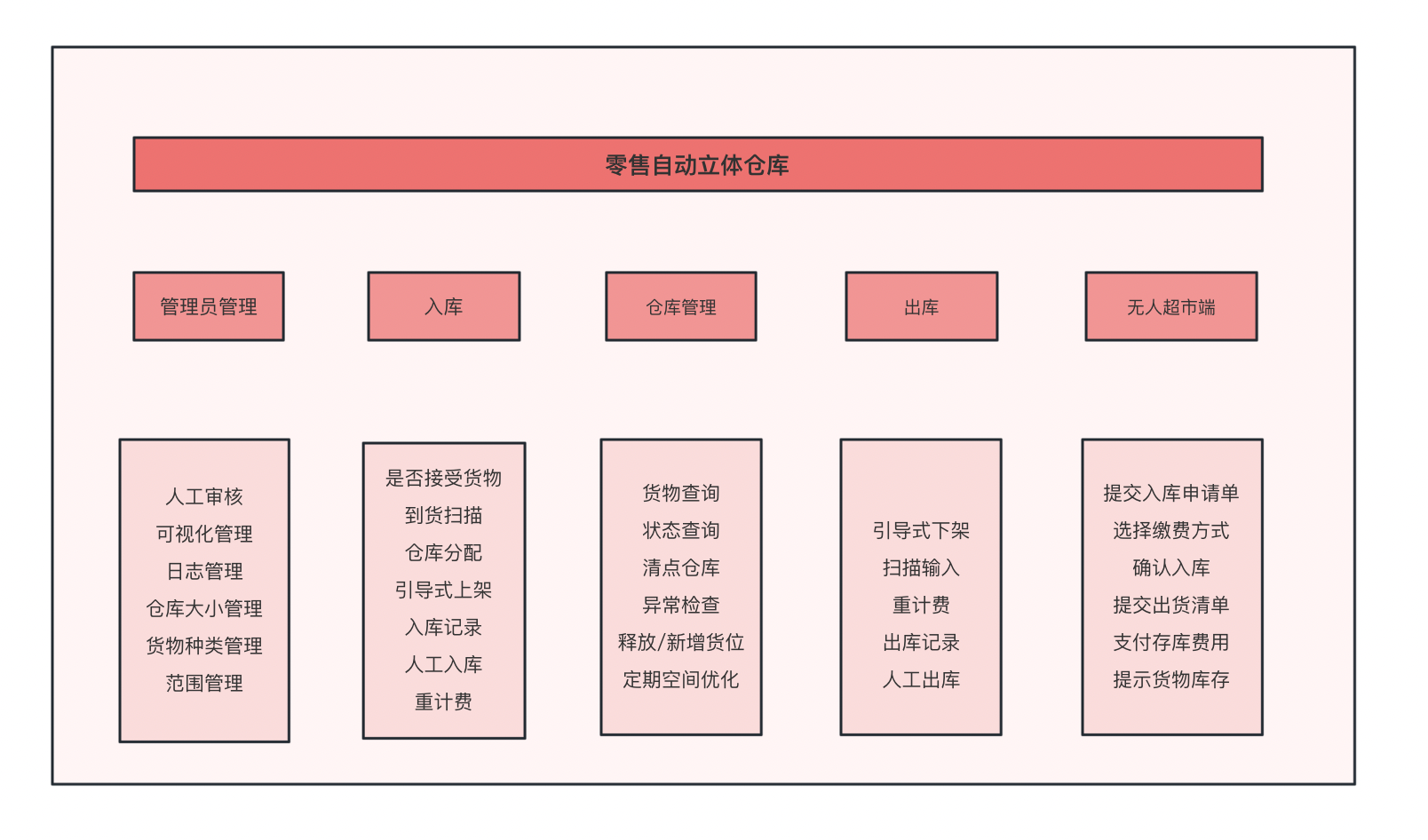
### **2.2.3微服务**

我们的后端项目结构从原先的SpringBoot+MyBatis框架的一个工程项目，扩展成了在SpringCloud框架的基础上的多个工程项目构成的微服务结构，共拆分成了六个模块，如下图所示。其中业务模块都存放在core文件夹中，现对各模块作用做详细阐述：

* admin服务：仓库库管服务。功能包括：审核入库申请单、查询仓库运作状态、查询出入库记录、查询仓库盈利统计、查询货物信息、查询仓库服务情况、仓库规则设置及查看操作日志等；
* supermarket服务：无人超市客户服务。功能包括：登录注册、提交入库申请单、选择缴费方式、查看缴费日志、缴费、审核核验单、提交出库申请单、查看订单状态日志和查询货物信息等；
* equipment服务：边缘设备控制服务。功能包括：模拟小车将货物运送到指定货位的调用接口，模拟货位上智能设备检查货物是否运送到正确位置的调用接口。值得说明的是，我们运用python程序来模拟边缘设备调用接口；
* pay服务：计费缴费服务。功能包括入库计费、出库重计费补缴费、通知无人超市客户选择缴费方式和缴费、返回缴费日志；
* eureka服务：实现微服务的注册与发现；
* gateway服务：实现路由映射等。



# 开发内容



# 架构设计的总结与思考

针对老师课堂上给出的意见即采用事件驱动机制完成入库工作流，我们从中学习到复杂的自动系统中的工作流完成消息自动传递并调用服务，可以采用kafka完成。微服务与微服务之间可耦合也可独立，入库的一步工作完成之后，可以将消息传入kafka通道中，监听的服务就会获取到消息信号，从而自动执行后续的过程，这样就实现了工作流机制，也就是事件驱动机制。

同时，在拆分微服务和加入授权与认证等技术的过程中，我们也踩了很多坑，积累了一些相关经验，学习到了如何构建微服务架构，服务与服务之间如何互相调用，如何形成事件驱动机制完成工作流等知识，更积累了完成大型工程项目的经验，受益匪浅。

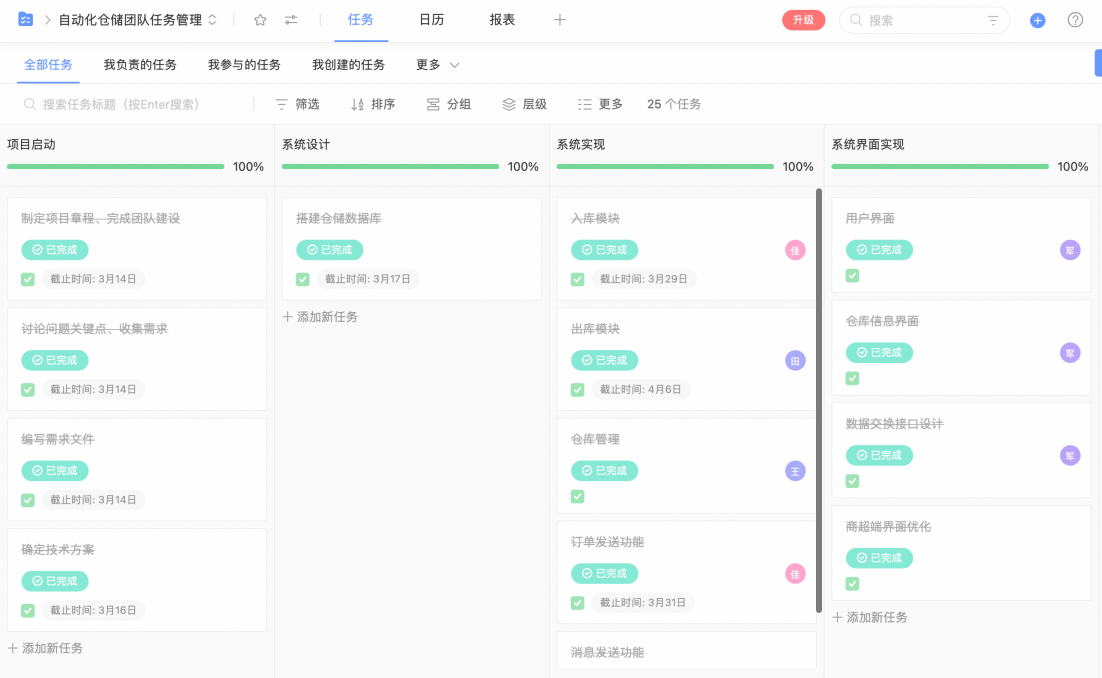
# 团队开发的总结与思考

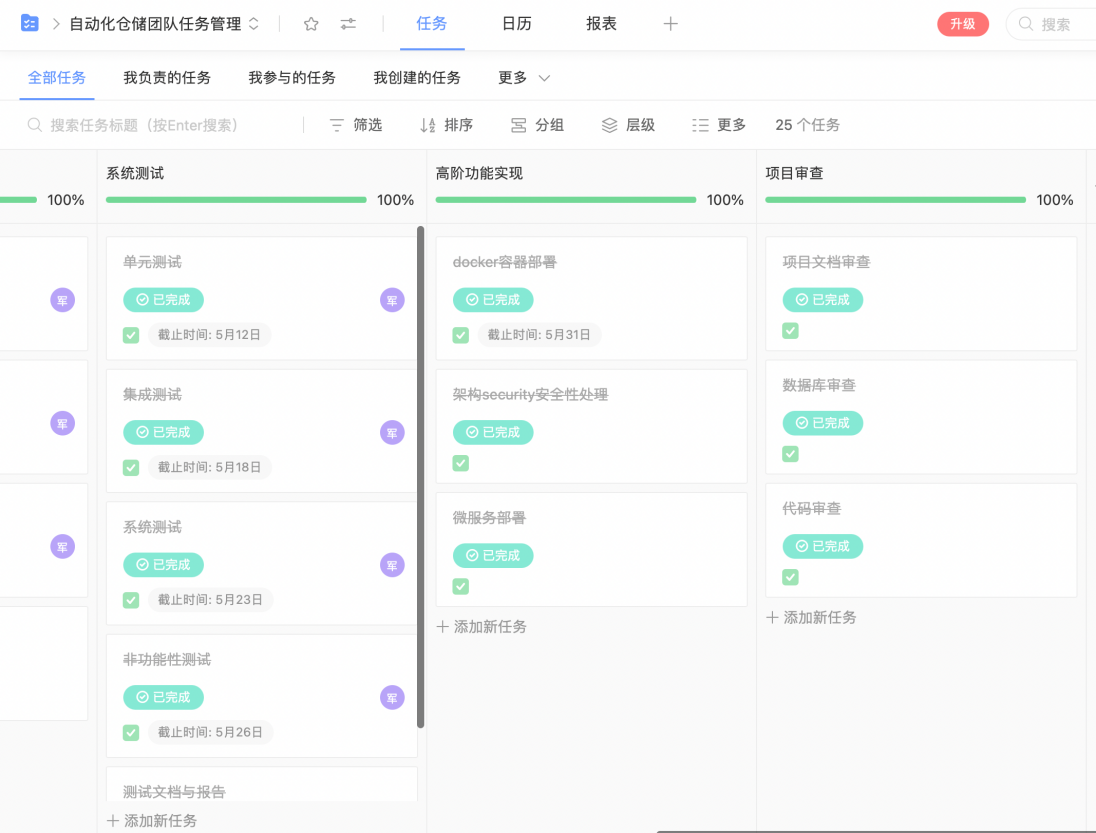
本次开发团队人员间已是第二次合作，在沟通与协作上很顺利。

小组成员贡献度：

|  |  |
| --- | --- |
| 成员（角色） | 贡献度占比 |
| 党田乐（组长、后端） | 21% |
| 何军庭（前端） | 19% |
| 王圳（后端） | 18% |
| 邹佳琪（后端） | 21% |
| 柳萱莹（前端） | 21% |

使用的项目管理工具是worktile：





# 代码管理的总结与思考

本次团队开发为前后端开发分离，分工为：后端3人（邹佳琪、党田乐、王圳），前端（柳萱莹、何军庭），前后端分别有一个github仓库，一个仓库中人员较少，因此使用一人一个分支的方法。

