项目设计文档

项目名称：GSP药店管理系统

学院：软件学院

专业：软件工程

班级：软工212

项目成员：

* 201周新斌
* 203纪润泽
* 210代萌
* 219薛薇
* 230郑语晖

# 一、系统建模

## 4.1 活动图

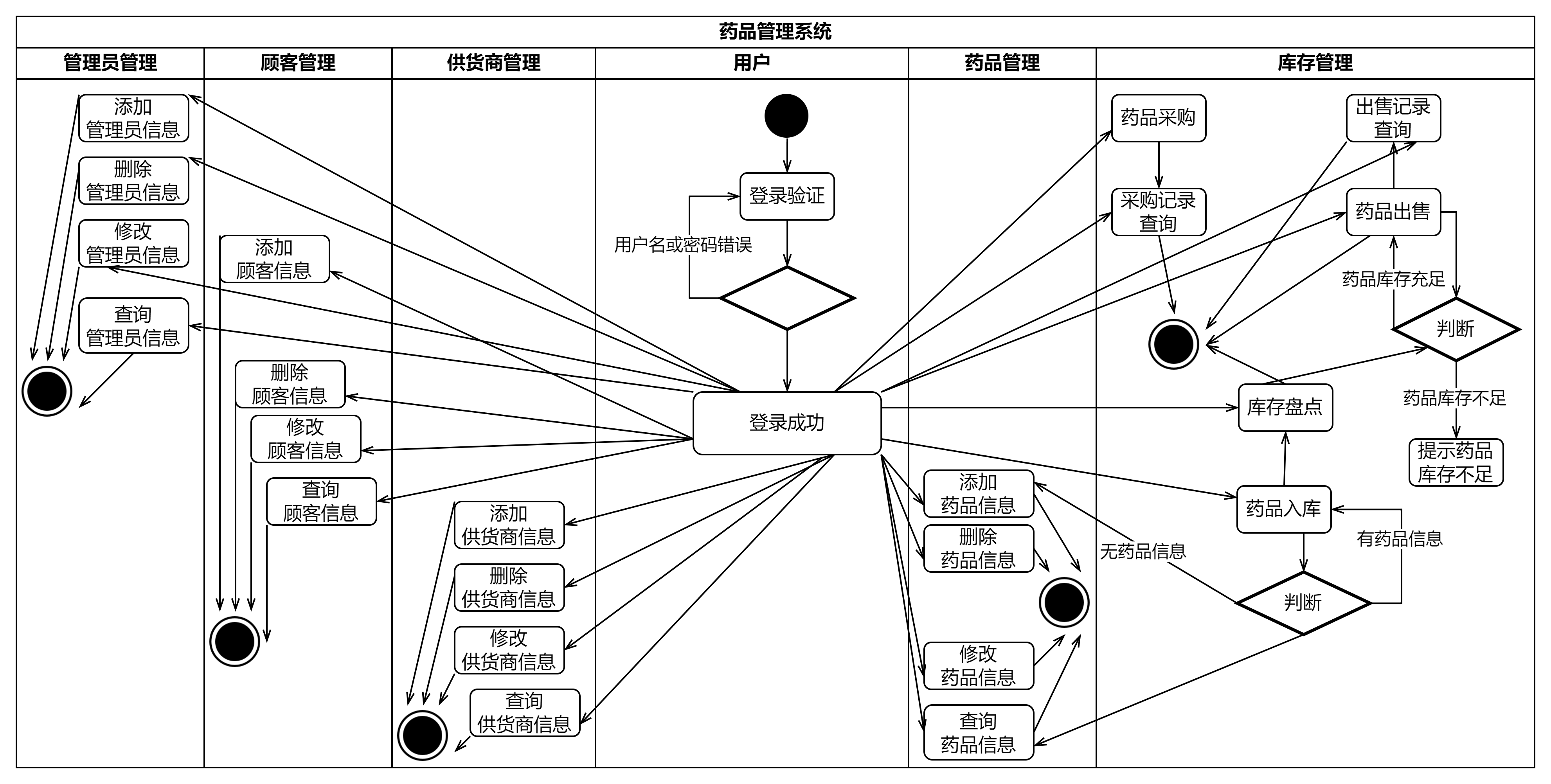


图4.1 活动图

## 4.2 时序图

### 4.2.1 售药

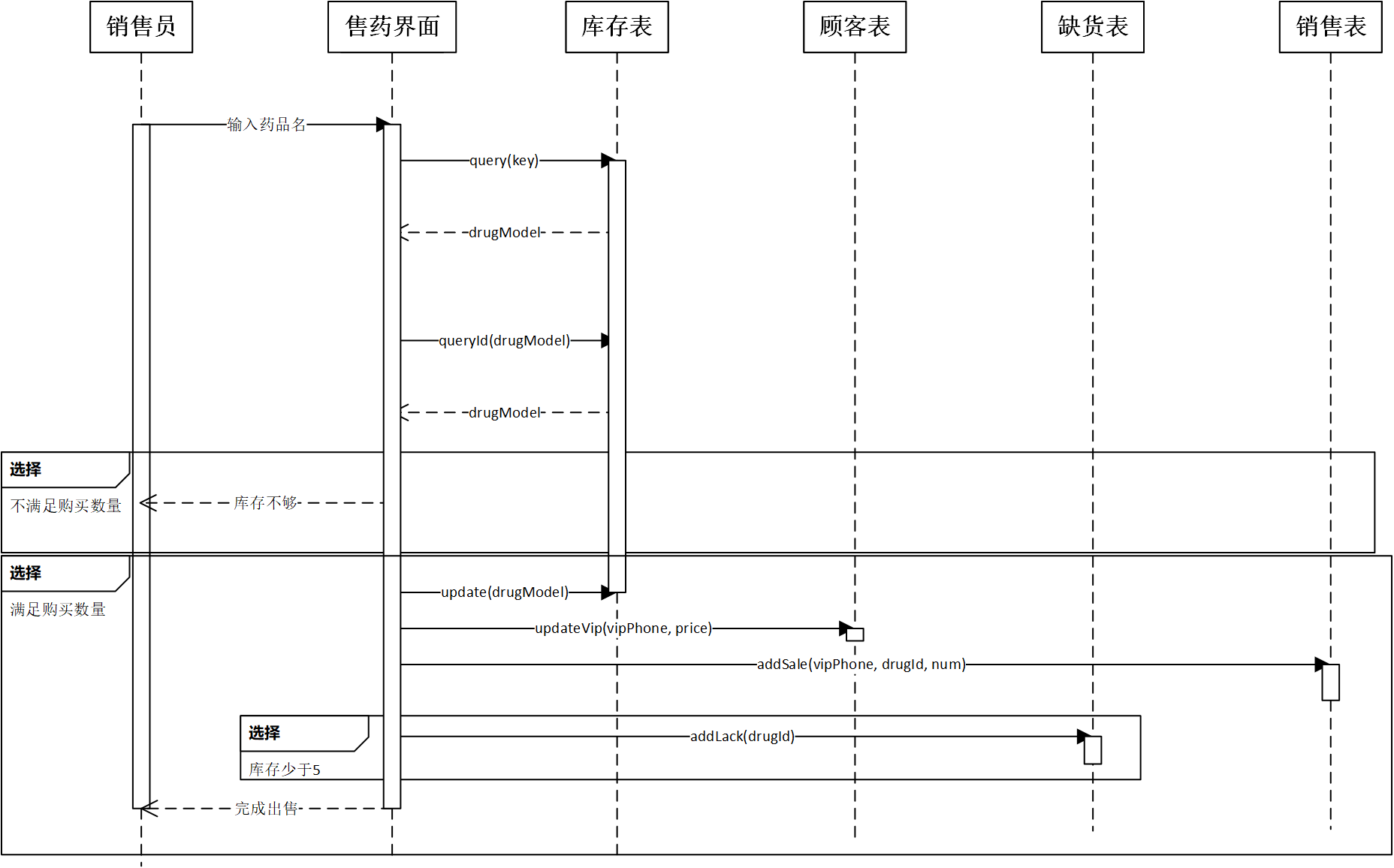
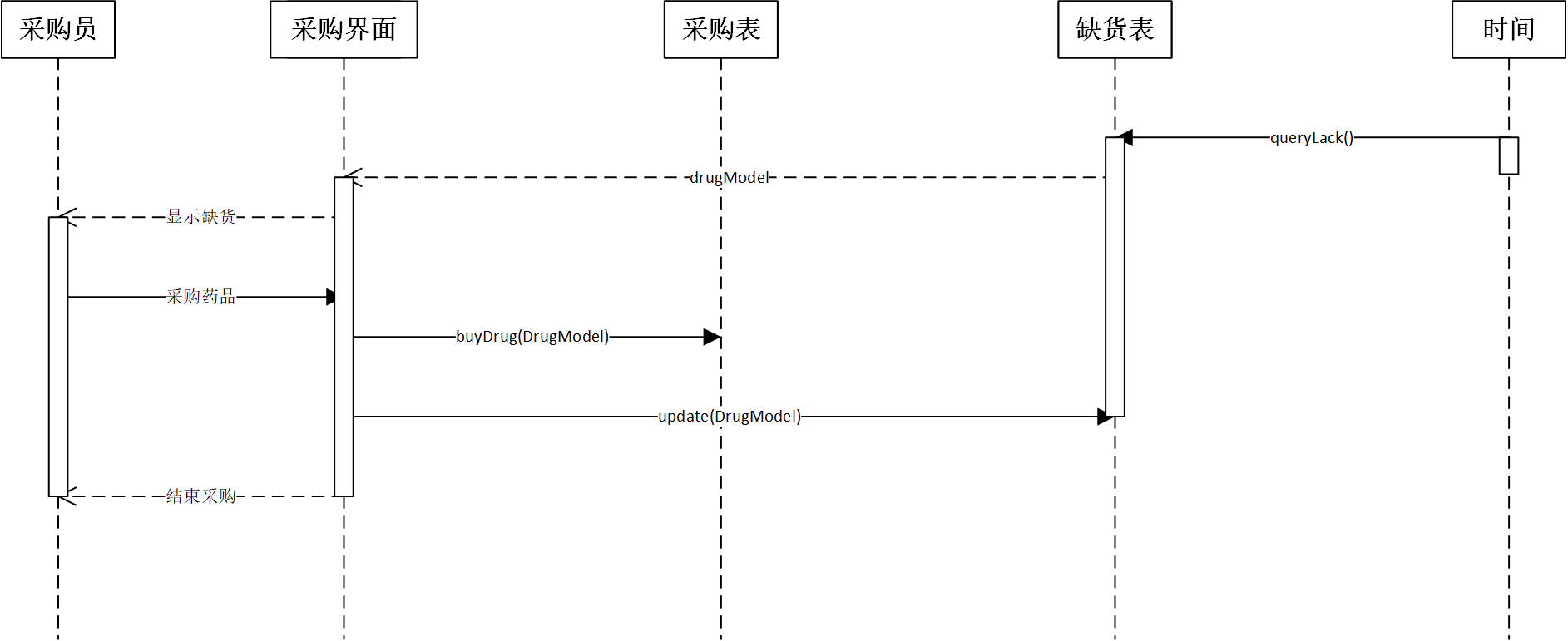


图4.2 售药时序图

### 4.2.2 采购入库



4.3 采购入库时序图

## 4.3 类图

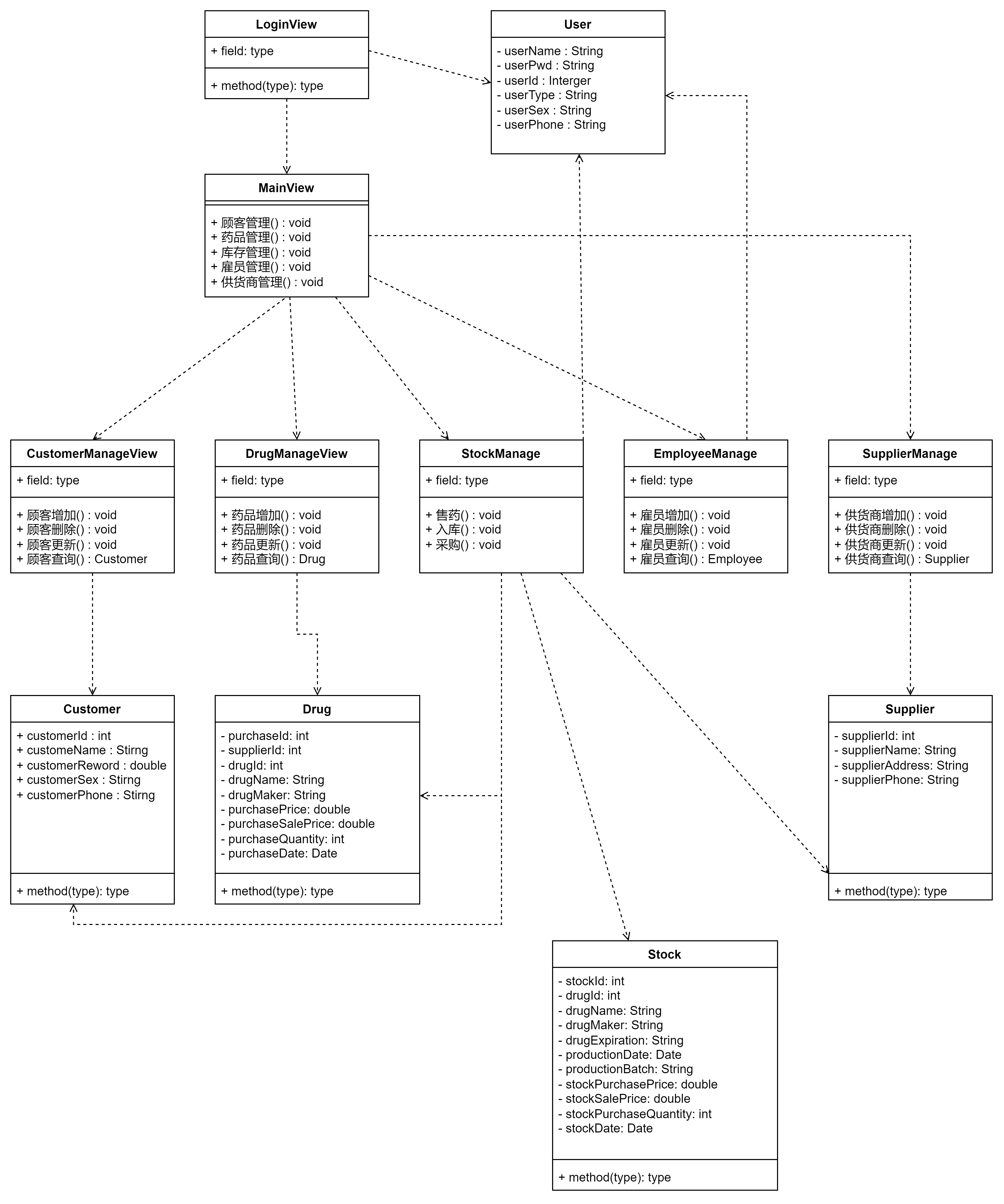


图4.4 类图

## 4.4 状态图

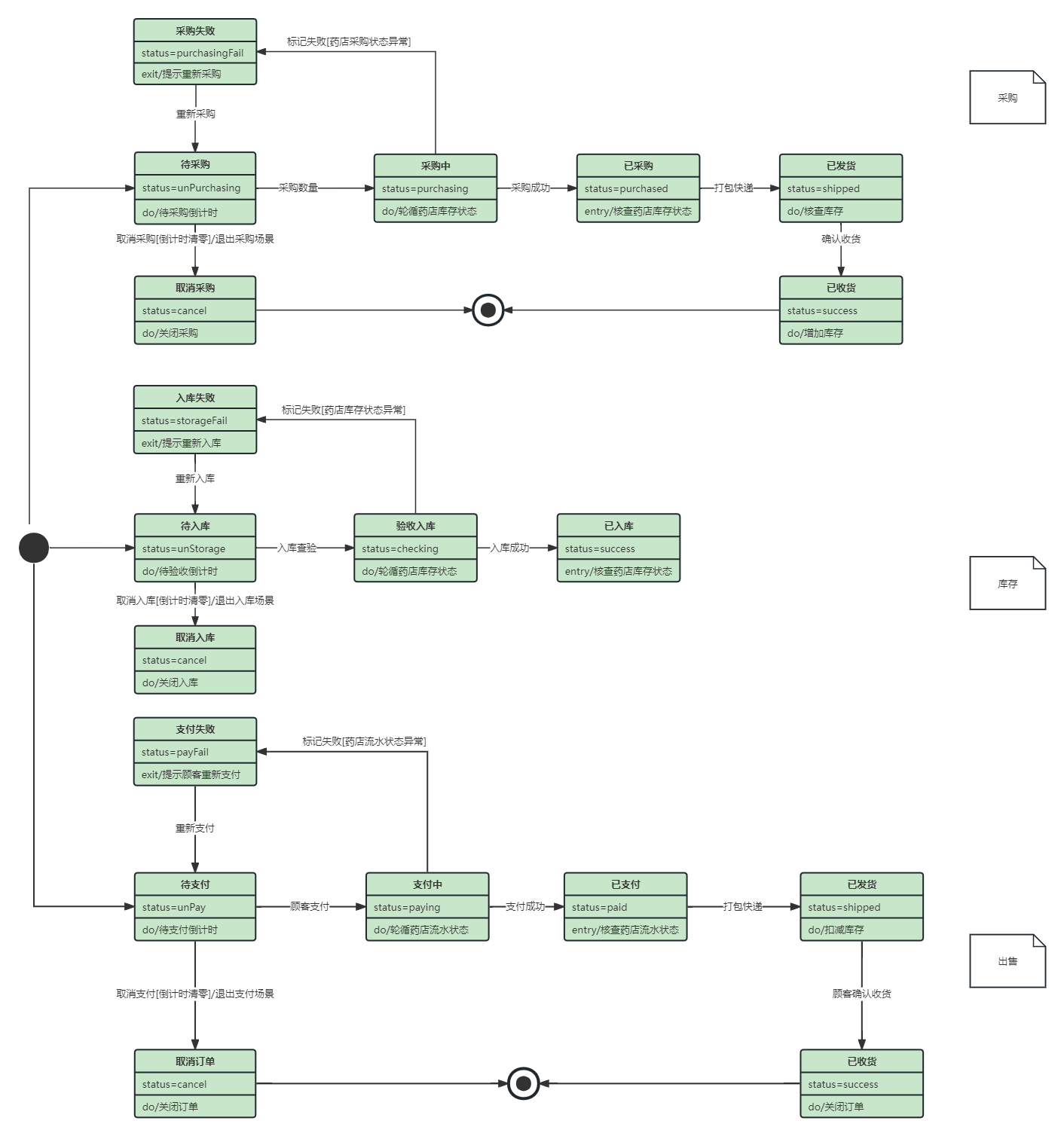


图4.5 状态图

# 二、系统体系架构

## 4.1 体系架构图

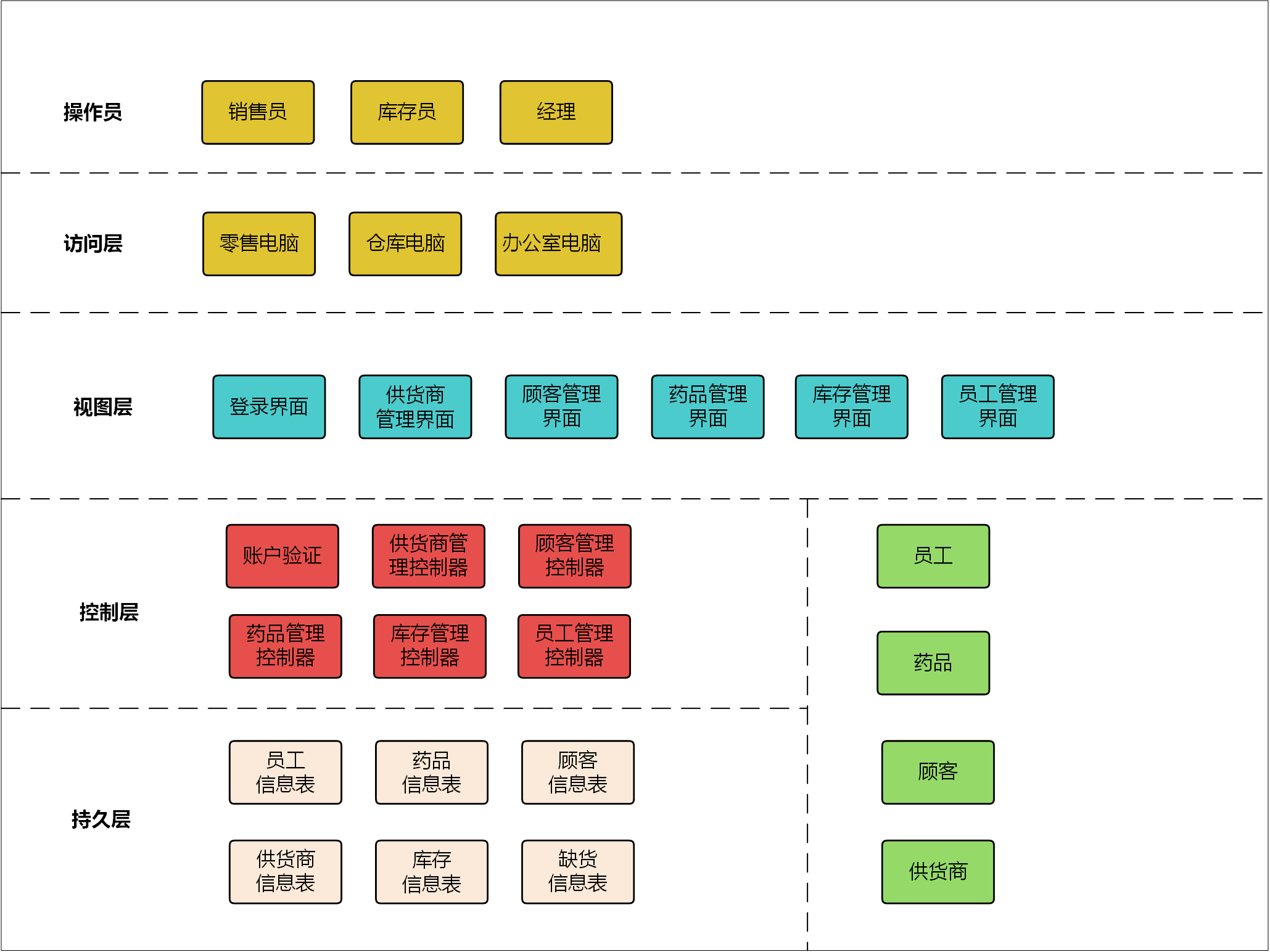


图5.1 系统体系架构图

1. 整体采用2层C/S架构。GSP药店管理系统业务简单，业务范围集中在某个药店内部，故采用2层C/S架构。业务操作在本地客户端执行，药店可以根据具体的本地需求定制和优化其界面和功能。通过向云端数据库服务器发送SQL语句进行读写操作。业务数据存放在云服务器可提高数据安全性和可恢复性。
2. 软件内部采用MVC架构。MVC架构通过分离数据模型（Model）、用户界面（View）和控制逻辑（Controller），实现了关注点的清晰划分。这种分离有助于减少组件间的依赖，使得开发、测试和维护各个部分变得更加简单和高效。
3. 数据库映射在药店管理系统中通过直接使用DOA在线映射实现，这种系统需求明确且功能相对简单，可以通过SQL语句直接与数据库进行交互。

## 4.2 核心代码实现

### 4.2.1 DBUtil类

在多个地方创建和配置数据库连接会增加系统的复杂性和运行开销。单例模式避免了这种情况，因为整个应用程序生命周期中只创建一个 **DBUtil** 实例。这减少了重复代码和对象创建所带来的开销，也使得对数据库工具类的修改更加集中和可控。

package SinbinZhou.Controller;  
import java.sql.\*;  
  
public class DBUtil {  
 private static DBUtil instance = new DBUtil();  
  
 private static String url = "jdbc:mysql://127.0.0.1:3306/javalab";  
 private static String username = "root";  
 private static String password = "";  
  
 private DBUtil() {  
 try {  
 Class.forName("com.mysql.cj.jdbc.Driver");  
 } catch (ClassNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 public static DBUtil getInstance() {  
 return instance;  
 }

public Connection getConnection() {  
 try {  
 return DriverManager.getConnection(url, username, password);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 // Static method to close the connection  
 public static void closeConnection(Connection connection) {  
 if (connection != null) {  
 try {  
 connection.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
  
 // Static method to close PreparedStatement  
 public static void closePs(PreparedStatement ps) {  
 if (ps != null) {  
 try {  
 ps.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
  
 // Static method to close ResultSet  
 public static void closeResultSet(ResultSet resultSet) {  
 if (resultSet != null) {  
 try {  
 resultSet.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
}

### 4.2.2 SalePartListener

public class SalePartListener implements ActionListener {  
 private SalePartView salePartView;  
 private String key;  
  
 // 构造函数，初始化销售部分视图  
 public SalePartListener(SalePartView salePartView) {  
 this.salePartView = salePartView;  
 }  
  
 // 实现ActionListener接口的方法，根据用户操作触发相应的处理  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource() == salePartView.getQueryProductButton()) {  
 performProductQuery(); // 执行产品查询  
 } else if(e.getSource() == salePartView.getSettlementButton()) {  
 handleProductSettlement(); // 处理产品结算  
 }  
 }  
  
 // 执行产品查询，显示查询结果  
 private void performProductQuery() {  
 key = salePartView.getQueryProductText().getText().trim();  
 if(key.isEmpty()) {  
 return;  
 }  
 MyTableModel myTableModel = SalePartController.query(key, new MyTableModel());  
 salePartView.getMyJTable().setMyTableModel(myTableModel);  
 }  
  
 // 处理产品结算，进行库存检查和计价  
 private void handleProductSettlement() {  
 String idStr = salePartView.getProductIDText().getText().trim();  
 String numStr = salePartView.getSealNumberText().getText().trim();  
 String vipPhone = salePartView.getVipPhoneText().getText.trim();  
  
 if (idStr.isEmpty() || numStr.isEmpty()) {  
 MyJOptionPane.showMessageDialog(null, "请输入商品ID和数量", "错误");  
 return;  
 }  
  
 try {  
 int id = Integer.parseInt(idStr);  
 int num = Integer.parseInt(numStr);  
 settleTransaction(id, num); // 结算交易  
 } catch (NumberFormatException ex) {  
 MyJOptionPane.showMessageDialog(null, "商品ID和数量必须是有效数字", "错误");  
 }  
 }  
  
 // 结算交易，更新库存和显示新价格  
 private void settleTransaction(int id, int num) {  
 ProductionModel temp = new ProductionModel();  
 temp.setId(id);  
 ProductionModel productionModel = SalePartController.idQuery(temp);  
 if (productionModel == null) {  
 MyJOptionPane.showMessageDialog(null, "未找到指定的商品", "错误");  
 return;  
 }  
  
 if (productionModel.getPurchaseQuantity() < num) {  
 MyJOptionPane.showMessageDialog(null, "库存不足", "提示");  
 return;  
 }  
  
 double totalPrice = num \* productionModel.getSalePrice();  
 salePartView.getTotalPriceText().setText("应收: " + formatPrice(totalPrice));  
 // 更新库存  
 productionModel.setPurchaseQuantity(productionModel.getPurchaseQuantity() - num);  
 SalePartController.update(productionModel);  
 // 添加销售记录  
 SalePartControllere.addSale(id, num, vipPhone);  
 // 更新积分  
 SalePartController.update(vipPhone, totalPrice);  
 // 更新查询列表  
 refreshQuery();  
 // 添加缺货记录  
 if (productionModel.getPurchaseQuantity() - num < 5) {  
 SalePartController.addLack(id); // 添加缺货记录  
 }  
 }  
  
 // 格式化价格显示  
 private String formatPrice(double price) {  
 return new DecimalFormat("#.##").format(price);  
 }  
  
 // 刷新查询显示，用于更新界面上的表格数据  
 private void refreshQuery() {  
 MyTableModel myTableModel = SalePartController.query(key, new MyTableModel());  
 salePartView.getMyJTable().setMyTableModel(myTableModel);  
 }  
}

### 4.2.3 SalePartContriller

public class SalePartController {  
 //模糊查询关键词, 不显示进价  
 public static MyTableModel query(String key, MyTableModel myTableModel) {  
 String sql = "SELECT id, name, factory, address, productionDate, " +  
 "expirationDate, purchaseQuantity, salePrice" +  
 " FROM product WHERE " +  
 "name LIKE '%" + key + "%' OR " +  
 "factory LIKE '%" + key + "%' OR " +  
 "address LIKE '%" + key + "%'";  
  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 Vector<Vector<Object>> data = new Vector<>();  
  
 try {  
 conn = DBUtil.getInstance().getConnection();  
 ps = conn.prepareStatement(sql);  
 rs = ps.executeQuery();  
 while(rs.next()) {  
 Vector<Object> dt = new Vector<>();  
 for (int i = 1; i <= 8; i++) {  
 dt.addElement(rs.getObject(i));  
 }  
 data.addElement(dt);  
 }  
 //向实体对象添加数据  
 myTableModel.setDate(data);  
 //向实体对象添加表头  
 Vector<Object> columns = new Vector<>();  
 columns.addElement("id");  
 columns.addElement("药品名称");  
 columns.addElement("生产厂家");  
 columns.addElement("生产地址");  
 columns.addElement("生产日期");  
 columns.addElement("有效期");  
 columns.addElement("库存数量");  
 columns.addElement("售价");  
 myTableModel.setColumns(columns);  
 //表格实体设置表头和数据  
 myTableModel.setDataVector(data, columns);  
 return myTableModel;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } finally {  
 DBUtil.closeResultSet(rs);  
 DBUtil.closePs(ps);  
 DBUtil.closeConnection(conn);  
 }  
 return null;  
 }  
  
 //根据id查询商品, 返回该id商品的库存和售价, 返回类型为Product  
 public static ProductionModel idQuery(ProductionModel productionModel) {  
 int qid = productionModel.getId();  
 String sql = "select purchaseQuantity, salePrice from product where " +  
 "id = " + qid + "";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
  
 try {  
 conn = DBUtil.getInstance().getConnection();  
 ps = conn.prepareStatement(sql);  
 rs = ps.executeQuery();  
 while(rs.next())  
 {  
 productionModel.setPurchaseQuantity(rs.getInt(1));  
 productionModel.setSalePrice(rs.getDouble(2));  
 }  
 return productionModel;  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 DBUtil.closeResultSet(rs);  
 DBUtil.closePs(ps);  
 DBUtil.closeConnection(conn);  
 }  
 return null;  
 }  
 //更新库存  
 public static void update(ProductionModel productionModel) {  
 String sql = "update product set purchaseQuantity = " +  
 productionModel.getPurchaseQuantity() + " " +  
 "where id = " + productionModel.getId() + "";  
 Connection conn = null;  
 PreparedStatement ps = null;  
  
 try {  
 conn = DBUtil.getInstance().getConnection();  
 ps = conn.prepareStatement(sql);  
 ps.executeUpdate();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 DBUtil.closePs(ps);  
 DBUtil.closeConnection(conn);  
 }  
 }  
 // 添加缺货日志  
 public static void addLack(int id)  
 {  
 String sql = "insert into lackLog (lackId, lackDate) values (?, ?)";  
 Connection conn = null;  
 PreparedStatement ps = null;  
 try {  
  
 conn = DBUtil.getInstance().getConnection();  
 ps = conn.prepareStatement(sql);  
 ps.setInt(1, id);  
 ps.setDate(2, new Date(System.currentTimeMillis()));  
 ps.executeUpdate();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 DBUtil.closePs(ps);  
 DBUtil.closeConnection(conn);  
 }  
 }  
}