# 实验二 智能制造系统订单管理模块设计与实现

# 1 EV-MES订单模块功能分析

## 1.1功能模块划分

**1.1.1订单基础管理模块**

（1）订单录入功能：支持新建订单，包含客户名称、车型、数量等基本信息录入。

（2）订单查询功能：提供订单列表展示，支持分页显示和客户名称模糊搜索。

（3）订单修改功能：支持编辑现有订单的所有字段信息。

（4）订单删除功能：提供订单删除操作，支持数据完整性保护

**1.1.2订单状态管理模块**

（1）状态跟踪功能：实现订单状态流转（新建→审核中→已完成）。

（2）状态更新功能：支持手动更新订单状态。

（3）状态统计功能：提供各状态订单数量统计和完成率计算。

**1.1.3订单可视化模块**

（1）统计卡片展示：显示总订单数、新建订单、审核中订单、已完成订单等关键指标。

（2）数据图表展示：以图表形式展示订单详细信息，方便查看数据。

（3）分页导航功能：提供分页控件，支持大量数据的高效浏览。

## 1.2系统功能图

如图1-1所示，本系统围绕订单全生命周期管理，从基础数据维护到状态追踪，再到多维度可视化呈现，构建了一个完整的订单管理生态。以订单基础管理模块为核心，实现对订单信息的基本维护；订单状态管理模块则通过状态跟踪与统计功能，呈现各阶段订单的流转情况；订单可视化模块借助卡片与图表展示方式，将关键业务指标直观呈现，提升效率。



图1-1 系统功能图

# 2 EV-MES订单管理功能流程图

如图2-1所示，系统以“录入—验证—审核—计划—完成”为主线，构建订单完整周期流程，层层校验、状态可视，确保数据准确和流程规范，有效提升订单处理效率与业务协同水平。

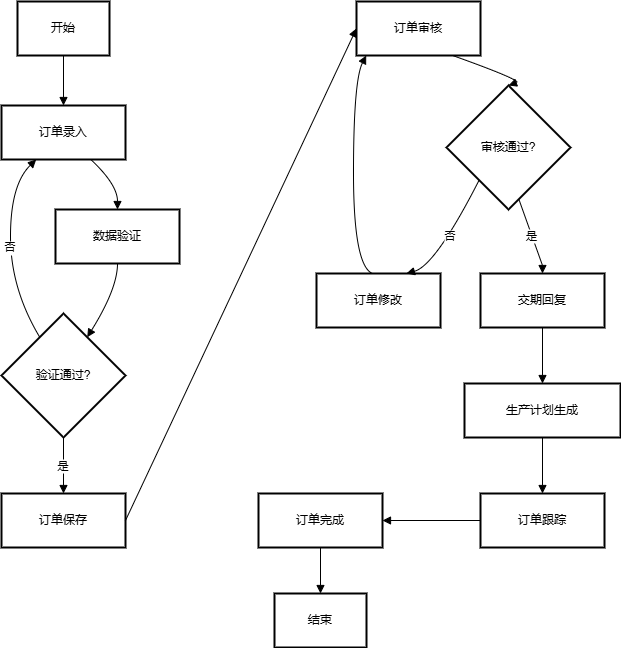


图2-1 功能流程图

# 3订单管理功能设计与实现

## 3.1 数据表设计

表3-1 orders表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 约束及含义 |
| id  customer | Integer  String(100) | 主键，自增  客户名称 |
| vehicle\_model  quantity  due\_date  status  vin\_prefix  created\_at  updated\_at | String(50)  Integer  DateTime  String(20)  String(10)  DateTime  DateTime | 车型  数量  交期  订单状态  VIN前缀  创建时间  更新时间 |

## 3.2 订单列表功能实现

（1）实现图

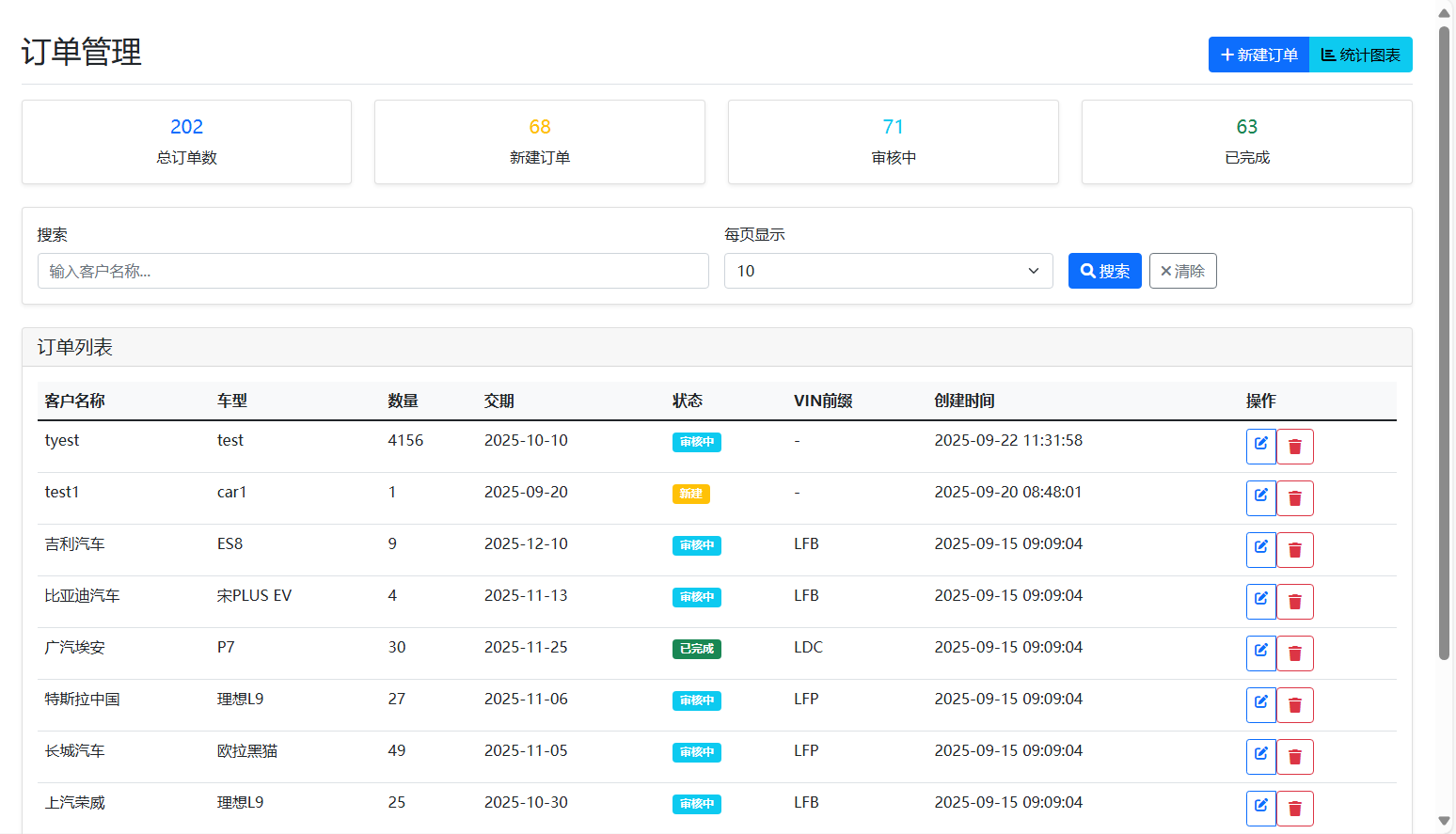


图3-1 订单列表功能

（2）核心代码

|  |
| --- |
| @order\_bp.route('/')  def page\_order\_list():  try:  db = session\_factory()  order\_service = OrderService(db)    page = int(request.args.get('page', 1))  per\_page = int(request.args.get('per\_page', 10))  search = request.args.get('search', '')    result = order\_service.get\_orders(page=page, per\_page=per\_page, search=search)    stats = order\_service.get\_order\_statistics()    return render\_template('order/list.html',  orders=result['orders'],  pagination=result,  search=search,  stats=stats,  status\_options=ORDER\_STATUS)  except Exception as e:  flash(f'获取订单列表失败: {str(e)}', 'error')  return render\_template('order/list.html', orders=[], pagination={}, search='', stats={}, status\_options=ORDER\_STATUS)  finally:  db.close() |

## 3.3 编辑订单功能实现

（1）实现图

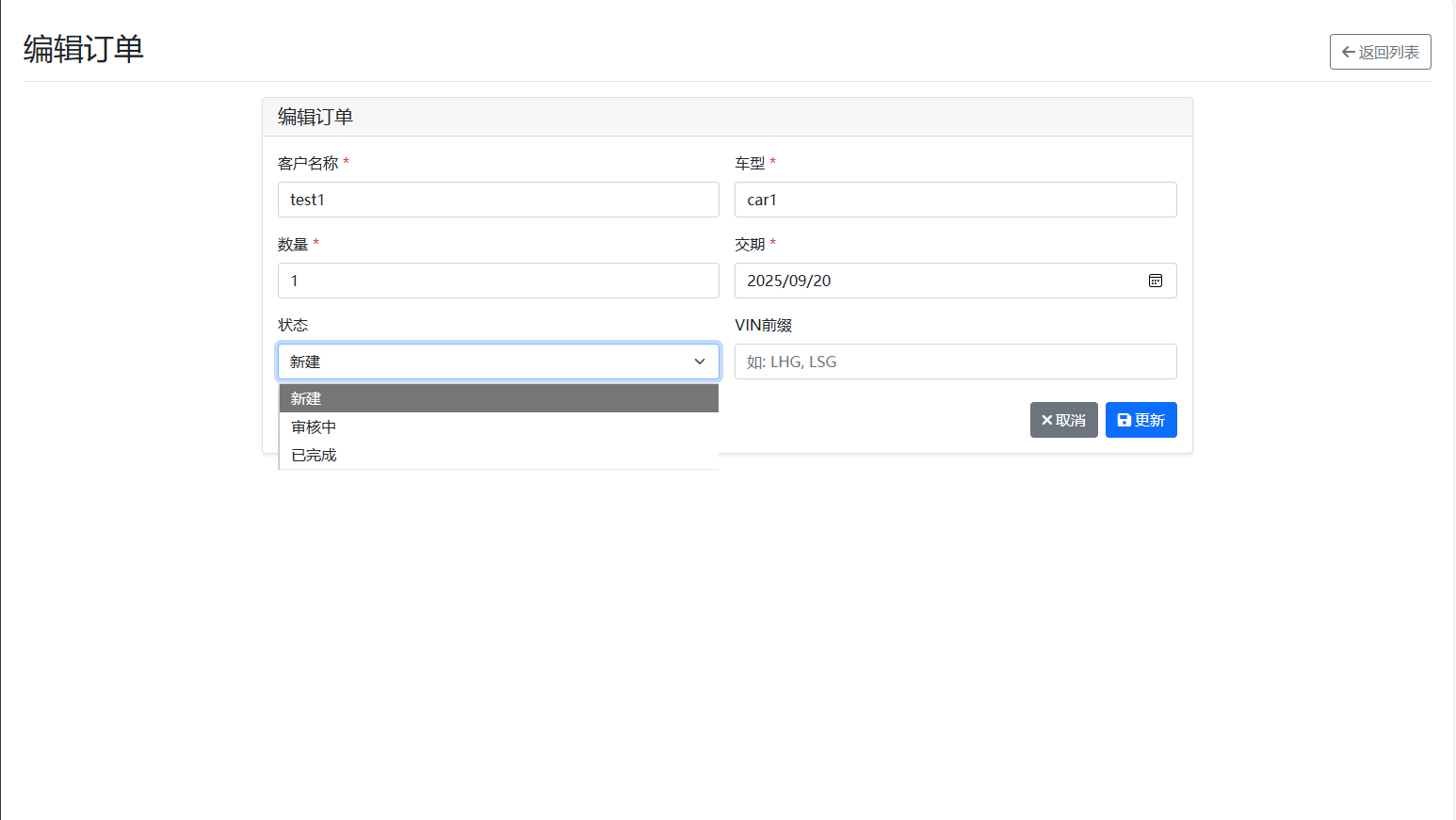


图3-2 编辑订单功能

（2）核心代码

|  |
| --- |
| def update\_order(self, order\_id: int, order\_data: Dict) -> Optional[Order]:  try:  order = self.get\_order\_by\_id(order\_id)  if not order:  return None    if 'customer' in order\_data:  order.customer = order\_data['customer']  if 'vehicle\_model' in order\_data:  order.vehicle\_model = order\_data['vehicle\_model']  if 'quantity' in order\_data:  order.quantity = order\_data['quantity']  if 'due\_date' in order\_data:  order.due\_date = datetime.strptime(order\_data['due\_date'], '%Y-%m-%d')  if 'status' in order\_data:  order.status = order\_data['status']  if 'vin\_prefix' in order\_data:  order.vin\_prefix = order\_data['vin\_prefix']  order.updated\_at = datetime.now()    self.db.commit()  self.db.refresh(order)    return order  except Exception as e:  self.db.rollback()  raise Exception(f"更新订单失败: {str(e)}") |

## 3.4 可视化统计功能实现

（1）实现图



图3-3 可视化统计功能

（2）核心代码

|  |
| --- |
| def create\_order\_charts(orders):  try:  if not orders:  return {'status\_chart': '', 'customer\_chart': ''}    status\_stats = {}  customer\_stats = {}    for order in orders:  status = order.get('status', '')  customer = order.get('customer', '')    if status:  if status not in status\_stats:  status\_stats[status] = 0  status\_stats[status] += 1    if customer:  if customer not in customer\_stats:  customer\_stats[customer] = 0  customer\_stats[customer] += 1    translated\_status\_stats = StatusMapping.translate\_order\_status\_dict(status\_stats)    top\_customers = dict(sorted(customer\_stats.items(), key=lambda x: x[1], reverse=True)[:10])    status\_chart = ''  customer\_chart = ''    if translated\_status\_stats:  status\_chart = MatplotlibCharts.create\_pie\_chart(  title='订单状态分布',  data=translated\_status\_stats  )    if top\_customers:  customer\_chart = MatplotlibCharts.create\_bar\_chart(  title='客户订单数量TOP10',  data=top\_customers  )    return {  'status\_chart': status\_chart,  'customer\_chart': customer\_chart  }    except Exception as e:  print(f"创建订单图表失败: {e}")  return {'status\_chart': '', 'customer\_chart': ''} |