MES 系统示例

一、数据库设计（MySQL）

1. 核心表结构

Code block

```sql
-- 1. 生产线表（基础数据）
CREATE TABLE production_line (
  line_id INT PRIMARY KEY AUTO_INCREMENT,
  line_name VARCHAR(50) NOT NULL COMMENT '生产线名称',
  status TINYINT DEFAULT 1 COMMENT '状态：1-运行中，0-停用',
  create_time DATETIME DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 2. 工单表（核心业务表）
CREATE TABLE work_order (
  order_id VARCHAR(32) PRIMARY KEY COMMENT '工单编号（如WO20251118001）',
  product_name VARCHAR(100) NOT NULL COMMENT '产品名称',
  plan_qty INT NOT NULL COMMENT '计划产量',
  actual_qty INT DEFAULT 0 COMMENT '实际产量',
  line_id INT COMMENT '所属生产线',
  status TINYINT DEFAULT 0 COMMENT '状态：0-待生产，1-生产中，2-已完成，3-暂停',
  start_time DATETIME COMMENT '开始时间',
  end_time DATETIME COMMENT '结束时间',
  create_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (line_id) REFERENCES production_line(line_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 3. 报工记录表（生产执行数据）
CREATE TABLE production_record (
  record_id INT PRIMARY KEY AUTO_INCREMENT,
  order_id VARCHAR(32) NOT NULL COMMENT '关联工单',
  qty INT NOT NULL COMMENT '本次报工数量',
  defective_qty INT DEFAULT 0 COMMENT '不良数量',
  operator VARCHAR(50) NOT NULL COMMENT '操作员',
  record_time DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '报工时间',
  FOREIGN KEY (order_id) REFERENCES work_order(order_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- 初始化测试数据
INSERT INTO production_line (line_name) VALUES ('SMT生产线1'), ('组装线A');
INSERT INTO work_order (order_id, product_name, plan_qty, line_id, status)
```

```sql
VALUES ('WO20251118001', '智能手机主板', 500, 1, 1);
```

二、后端代码（Spring Boot）

1. 项目结构

Code block

```
com.mes
├── controller       // 接口层
│   └── WorkOrderController.java
├── entity           // 实体类
│   ├── WorkOrder.java
│   └── ProductionRecord.java
├── repository       // 数据访问层（MyBatis Mapper）
│   └── WorkOrderMapper.java
├── service          // 业务层
│   ├── WorkOrderService.java
│   └── impl/WorkOrderServiceImpl.java
├── dto              // 数据传输对象
│   ├── OrderQueryDTO.java
│   └── ProductionRecordDTO.java
└── MesApplication.java  // 启动类
```

2. 核心代码

（1）实体类 `WorkOrder.java`

Code block

```java
package com.mes.entity;

import java.time.LocalDateTime;
import lombok.Data;

@Data
public class WorkOrder {
    private String orderId;
    private String productName;
    private Integer planQty;
    private Integer actualQty;
    private Integer lineId;
    private Integer status; // 0-待生产，1-生产中，2-已完成，3-暂停
    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private LocalDateTime createTime;
```

```
}
```

（2）Mapper接口 `WorkOrderMapper.java` （MyBatis）

Code block

```
package com.mes.repository;

import com.mes.entity.WorkOrder;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import java.util.List;

@Mapper
public interface WorkOrderMapper {
    // 查询工单列表
    List<WorkOrder> selectByCondition(@Param("status") Integer status,
@Param("lineId") Integer lineId);

    // 更新工单状态
    int updateStatus(@Param("orderId") String orderId, @Param("status")
Integer status);

    // 新增报工记录
    int insertProductionRecord(@Param("record") ProductionRecord record);

    // 更新工单实际产量
    int updateActualQty(@Param("orderId") String orderId, @Param("addQty")
Integer addQty);
}
```

（3）Service实现 `WorkOrderServiceImpl.java`

Code block

```
package com.mes.service.impl;

import com.mes.entity.ProductionRecord;
import com.mes.entity.WorkOrder;
import com.mes.repository.WorkOrderMapper;
import com.mes.service.WorkOrderService;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import javax.annotation.Resource;
import java.time.LocalDateTime;
import java.util.List;
```

```java

@Service
public class WorkOrderServiceImpl implements WorkOrderService {

    @Resource
    private WorkOrderMapper workOrderMapper;

    @Override
    public List<WorkOrder> getOrderList(Integer status, Integer lineId) {
        return workOrderMapper.selectByCondition(status, lineId);
    }

    @Transactional // 事务保证：报工记录和产量更新同时成功/失败
    @Override
    public boolean reportProduction(ProductionRecord record) {
        // 1. 新增报工记录
        workOrderMapper.insertProductionRecord(record);
        // 2. 更新工单实际产量
        workOrderMapper.updateActualQty(record.getOrderId(), record.getQty());
        // 3. 如果实际产量达到计划产量，自动标记为完成
        WorkOrder order = workOrderMapper.selectById(record.getOrderId());
        if (order.getActualQty() >= order.getPlanQty()) {
            workOrderMapper.updateStatus(record.getOrderId(), 2); // 2-已完成
            order.setEndTime(LocalDateTime.now());
        }
        return true;
    }
}
```

(4) Controller `WorkOrderController.java`

Code block

```java
package com.mes.controller;

import com.mes.dto.ProductionRecordDTO;
import com.mes.entity.WorkOrder;
import com.mes.service.WorkOrderService;
import org.springframework.web.bind.annotation.*;
import javax.annotation.Resource;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/workorder")
public class WorkOrderController {
```

```java

    @Resource
    private WorkOrderService workOrderService;

    // 获取工单列表
    @GetMapping("/list")
    public Map<String, Object> getOrderList(
            @RequestParam(required = false) Integer status,
            @RequestParam(required = false) Integer lineId) {
        List<WorkOrder> list = workOrderService.getOrderList(status, lineId);
        Map<String, Object> result = new HashMap<>();
        result.put("code", 200);
        result.put("data", list);
        return result;
    }

    // 生产报工
    @PostMapping("/report")
    public Map<String, Object> reportProduction(@RequestBody
ProductionRecordDTO dto) {
        boolean success = workOrderService.reportProduction(dto.toEntity());
        Map<String, Object> result = new HashMap<>();
        result.put("code", success ? 200 : 500);
        result.put("message", success ? "报工成功" : "报工失败");
        return result;
    }
}
```

三、前端代码（Vue 3 + Element Plus）

1. 项目结构（核心部分）

Code block

```
src/
├── views/
│   └── WorkOrder/
│       ├── OrderList.vue       // 工单列表页面
│       └── ProductionReport.vue // 生产报工页面
├── api/
│   └── workorder.js            // 接口请求封装
└── main.js
```

2. 核心代码

（1）接口封装 `api/workorder.js`

Code block

```
import axios from 'axios'

// 基础路径（实际项目中配置环境变量）
const baseURL = 'http://localhost:8080/api'

// 获取工单列表
export const getOrderList = (params) => {
  return axios.get(`${baseURL}/workorder/list`, { params })
}

// 生产报工
export const reportProduction = (data) => {
  return axios.post(`${baseURL}/workorder/report`, data)
}
```

（2）工单列表页面 `OrderList.vue`

Code block

```
<template>
  <div class="order-list">
    <el-card>
      <div class="filter-bar">
        <el-select v-model="status" placeholder="工单状态" clearable>
          <el-option label="待生产" value="0"></el-option>
          <el-option label="生产中" value="1"></el-option>
          <el-option label="已完成" value="2"></el-option>
        </el-select>
        <el-button type="primary" @click="fetchOrderList">查询</el-button>
      </div>

      <el-table :data="orderList" border>
        <el-table-column prop="orderId" label="工单编号"></el-table-column>
        <el-table-column prop="productName" label="产品名称"></el-table-column>
        <el-table-column prop="planQty" label="计划产量"></el-table-column>
        <el-table-column prop="actualQty" label="实际产量"></el-table-column>
        <el-table-column prop="status" label="状态">
          <template #default="scope">
            <el-tag :type="statusMap[scope.row.status].type">
              {{ statusMap[scope.row.status].label }}
            </el-tag>
          </template>
```

```
          </el-table-column>
          <el-table-column label="操作">
            <template #default="scope">
              <el-button
                type="success"
                size="small"
                @click="handleReport(scope.row)"
                :disabled="scope.row.status === 2"
              >
                报工
              </el-button>
            </template>
          </el-table-column>
        </el-table>
      </el-card>

      <!-- 报工弹窗 -->
      <production-report
        :visible="reportVisible"
        :current-order="currentOrder"
        @close="reportVisible = false"
        @success="onReportSuccess"
      ></production-report>
    </div>
  </template>

  <script setup>
  import { ref, onMounted } from 'vue'
  import { getOrderList } from '@/api/workorder'
  import ProductionReport from './ProductionReport.vue'

  // 状态映射
  const statusMap = {
    0: { label: '待生产', type: 'warning' },
    1: { label: '生产中', type: 'primary' },
    2: { label: '已完成', type: 'success' },
    3: { label: '暂停', type: 'danger' }
  }

  // 响应式数据
  const orderList = ref([])
  const status = ref('')
  const reportVisible = ref(false)
  const currentOrder = ref(null)

  // 获取工单列表
  const fetchOrderList = async () => {
```

```
  const res = await getOrderList({ status: status.value })
  if (res.data.code === 200) {
    orderList.value = res.data.data
  }
}

// 打开报工弹窗
const handleReport = (row) => {
  currentOrder.value = row
  reportVisible.value = true
}

// 报工成功后刷新列表
const onReportSuccess = () => {
  fetchOrderList()
}

// 页面加载时查询
```

（3）报工页面 `ProductionReport.vue`

Code block

```
<template>
  <el-dialog
    title="生产报工"
    v-model="visible"
    width="400px"
    @close="handleClose"
  >
    <el-form :model="form" label-width="100px">
      <el-form-item label="工单编号" disabled>
        <el-input v-model="form.orderId"></el-input>
      </el-form-item>
      <el-form-item label="产品名称" disabled>
        <el-input v-model="form.productName"></el-input>
      </el-form-item>
      <el-form-item label="本次产量" prop="qty" :rules="[{ required: true, message: '请输入产量', type: 'number' }]">
        <el-input v-model.number="form.qty" type="number" min="1"></el-input>
      </el-form-item>
      <el-form-item label="不良数量">
        <el-input v-model.number="form.defectiveQty" type="number" min="0"></el-input>
      </el-form-item>
      <el-form-item label="操作员">
        <el-input v-model="form.operator"></el-input>
      </el-form-item>
```

```
      </el-form>
      <template #footer>
        <el-button @click="visible = false">取消</el-button>
        <el-button type="primary" @click="submitReport">提交</el-button>
      </template>
    </el-dialog>
  </template>

  <script setup>
  import { ref, defineProps, defineEmits } from 'vue'
  import { reportProduction } from '@/api/workorder'

  // 接收父组件参数
  const props = defineProps({
    visible: Boolean,
    currentOrder: Object
  })

  // 触发父组件事件
  const emit = defineEmits(['close', 'success'])

  // 表单数据
  const form = ref({
    orderId: '',
    productName: '',
    qty: 1,
    defectiveQty: 0,
    operator: ''
  })

  // 监听currentOrder变化，初始化表单
  watch(() => props.currentOrder, (val) => {
    if (val) {
      form.value.orderId = val.orderId
      form.value.productName = val.productName
    }
  })

  // 提交报工
  const submitReport = async () => {
    try {
      const res = await reportProduction(form.value)
      if (res.data.code === 200) {
        ElMessage.success('报工成功')
        emit('success')
        emit('close')
      } else {
```

```
        ElMessage.error(res.data.message)
      }
    } catch (error) {
      ElMessage.error('网络错误')
    }
  }

  // 关闭弹窗时重置表单
  const handleClose = () => {
    form.value = {
      orderId: '',
      productName: '',
      qty: 1,
      defectiveQty: 0,
      operator: ''
    }
```

四、运行说明

1. **后端**：
   - 创建Spring Boot项目，引入依赖（Spring Web、MyBatis、MySQL Driver）。
   - 配置 `application.yml`（数据库连接、MyBatis映射路径）。
   - 启动类添加 `@MapperScan("com.mes.repository")`。
2. **前端**：
   - 创建Vue项目：`npm create vite@latest mes-frontend -- --template vue`。
   - 安装依赖：`npm install element-plus axios`。
   - 启动：`npm run dev`。
3. **测试流程**：
   - 访问前端工单列表页面，加载初始化的工单数据。
   - 点击“报工”按钮，输入产量和操作员信息，提交后可看到实际产量更新。

扩展建议

1. 增加权限控制（Spring Security + JWT）。
2. 集成WebSocket实现工单状态实时刷新。
3. 扩展设备管理模块（设备状态监控、维护记录）。
4. 加入数据看板（用ECharts展示产量趋势、合格率等）。

（注：文档部分内容可能由 AI 生成）