# 运输web系统设计报告

## 1. 系统设计目的

本系统旨在提供一个可靠、高效的管理和跟踪货物的解决方案。通过该系统，用户可以方便地创建、分配、跟踪和管理运输货物，实现运输过程的自动化和优化。

## 2. 系统架构设计

### 2.1 总体架构

在总体架构上，入库单子系统采用以下组件和层次结构：

* 用户界面层：提供用户与系统交互的界面，位于com.example.warehouse.controller包下，负责接收用户的请求并将其转发给应用服务层进行处理。
* 应用服务层：处理用户请求，协调各个模块之间的交互，负责业务逻辑的处理。位于com.example.warehouse.service包下，包含各种服务和业务组件，用于实现货物的创建、查询、分配、状态更新等功能。应用服务层通过调用数据访问层提供的接口与数据库进行交互，并将结果返回给用户界面层。
* 数据访问层：负责与数据库交互，进行数据的读取和存储。位于com.example.warehouse.mapper包下，使用数据访问技术（MyBatis）来执行数据库操作。数据访问层包含与数据库交互的接口和实现，用于实现对入库单数据的读写操作。
* 数据库层：持久化存储入库单相关的数据，系统使用关系型数据库（MySQL），数据库层的配置信息在application.yaml文件中进行配置。

总体架构的设计遵循了典型的三层架构模式，将系统的不同功能和责任进行了合理划分。

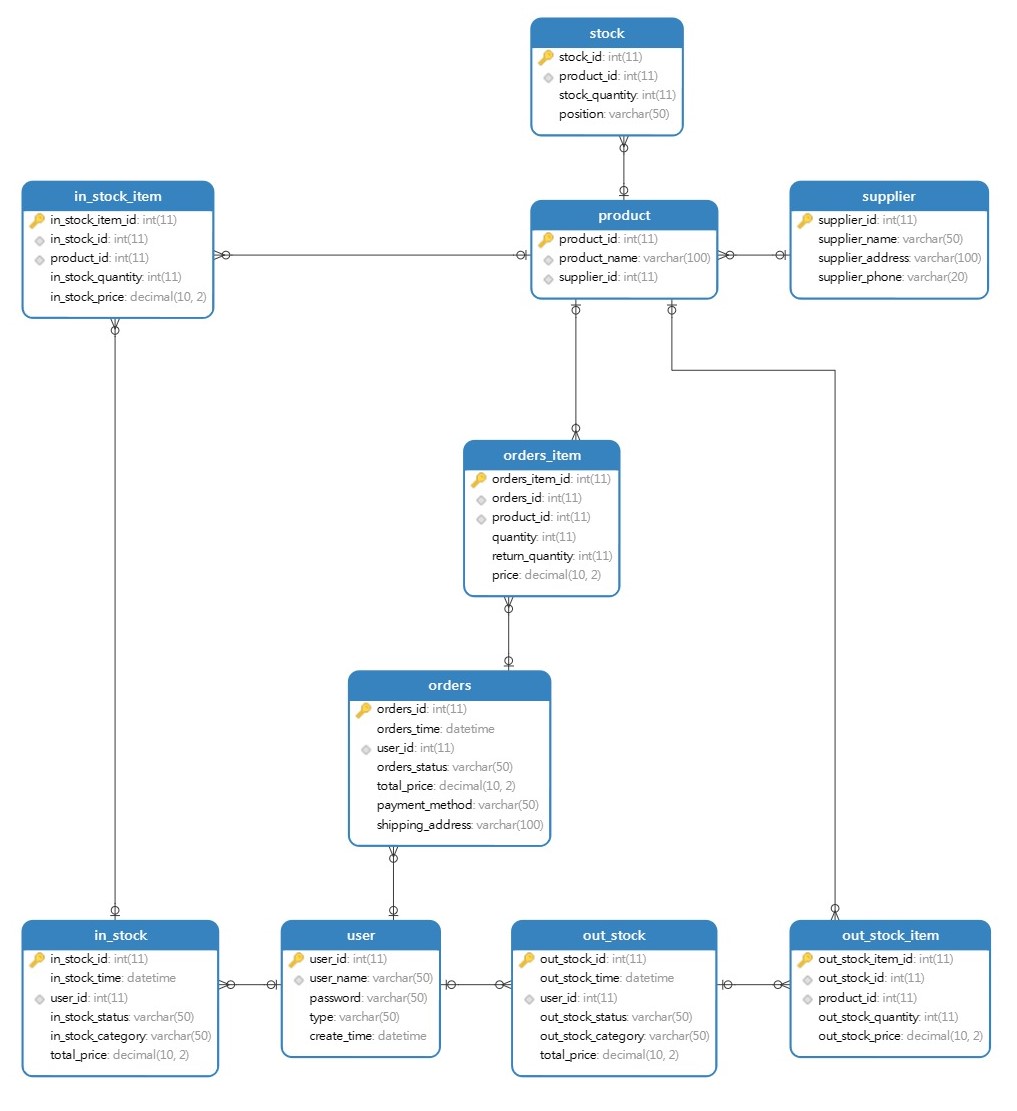
用户界面层、应用服务层、数据访问层之间通过定义良好的接口进行交互，实现了模块之间的解耦和灵活性。

### 2.2 主要组件和模块

* 用户界面模块：负责提供用户友好的界面，包括货物创建、查询、分配、状态更新等操作的界面。
* 货物管理模块：处理货物清单的创建、更新、查询、分配和取消等操作。
* 调度管理模块：负责自动分配小车进行调度和手动分配调度的管理。
* 数据库管理模块：处理与数据库的交互，包括入库单数据的读取和存储。

## 4. 数据库设计

以下为整个系统对应的数据库，和入库单子系统相关的表为in\_stock、in\_stock\_item、product



### 入库表

CREATE TABLE in\_stock (  
 in\_stock\_id INT NOT NULL AUTO\_INCREMENT,  
 in\_stock\_time DATETIME NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 user\_id INT,  
 in\_stock\_status VARCHAR(50),  
 in\_stock\_category VARCHAR(50),  
 PRIMARY KEY (in\_stock\_id),  
 CONSTRAINT fk\_user\_in\_stock FOREIGN KEY (user\_id) REFERENCES user (user\_id) ON DELETE CASCADE ON UPDATE CASCADE  
);

上述表结构包括以下字段：

* in\_stock\_id：入库单唯一标识符，自动递增生成的整数。
* in\_stock\_time：入库时间，存储日期和时间的信息。
* user\_id：用户ID，与用户表中的用户ID相关联。
* in\_stock\_status：入库状态，存储为字符串。
* in\_stock\_category：入库类别，存储为字符串。

示例数据：

INSERT INTO `in\_stock` VALUES (2, '2023-04-20 11:31:56', 1, 'uncompleted', 'production', 3187000.00);

### 入库商品明细表

CREATE TABLE in\_stock\_item (  
 in\_stock\_item\_id INT NOT NULL AUTO\_INCREMENT,  
 in\_stock\_id INT,  
 product\_id INT,  
 in\_stock\_quantity INT,  
 in\_stock\_price DECIMAL(10, 2),  
 PRIMARY KEY (in\_stock\_item\_id),  
 CONSTRAINT fk\_in\_stock FOREIGN KEY (in\_stock\_id) REFERENCES in\_stock (in\_stock\_id) ON DELETE CASCADE ON UPDATE CASCADE,  
 CONSTRAINT fk\_product\_in\_stock FOREIGN KEY (product\_id) REFERENCES product (product\_id) ON DELETE CASCADE ON UPDATE CASCADE  
);

上述表结构包括以下字段：

* in\_stock\_item\_id：入库商品明细唯一标识符，自动递增生成的整数。
* in\_stock\_id：入库单ID，与在库表中的入库单ID相关联。
* product\_id：商品ID，与商品表中的商品ID相关联。
* in\_stock\_quantity：入库数量，存储整数值。
* in\_stock\_price：入库价格，存储为带有两位小数的小数值。

示例数据：

INSERT INTO `in\_stock\_item` VALUES (1, 1, 1, 999, 2499.00);  
INSERT INTO `in\_stock\_item` VALUES (2, 1, 2, 1000, 4999.00);

### 商品表

CREATE TABLE product (  
 product\_id INT NOT NULL AUTO\_INCREMENT,  
 product\_name VARCHAR(100) UNIQUE,  
 supplier\_id INT,  
 PRIMARY KEY (product\_id),  
 CONSTRAINT fk\_supplier FOREIGN KEY (supplier\_id) REFERENCES supplier (supplier\_id) ON DELETE CASCADE ON UPDATE CASCADE  
);

上述表结构包括以下字段：

* product\_id：商品唯一标识符，自动递增生成的整数。
* product\_name：商品名称，存储为字符串，具有唯一性约束。
* supplier\_id：供应商ID，与供应商表中的供应商ID相关联。

示例数据：

INSERT INTO `product` VALUES (3, 'iPhone 13', 1);  
INSERT INTO `product` VALUES (12, 'HUAWEI P30', 2);

## 5.安全性设计

1. 访问控制：确保只有经过身份验证和授权的用户能够访问系统的敏感功能和数据。可以使用身份验证和授权框架（如Spring Security）来实现用户认证和授权管理。
2. 数据加密：对于敏感数据（如用户凭据、交易信息等），应使用适当的加密算法进行加密，确保数据在存储和传输过程中的安全性。可以使用TLS/SSL协议来保护数据在网络传输中的安全性。
3. 输入验证：对于用户输入的数据，进行有效性验证和过滤，以防止潜在的安全漏洞，如SQL注入、跨站脚本攻击（XSS）等。可以使用输入验证框架（如Hibernate Validator）来验证用户输入的数据。
4. 强密码策略：对于用户账户密码，应强制要求使用强密码，并采用适当的密码存储方案（如使用哈希算法和盐值存储密码），确保密码的安全性。
5. 日志记录和审计：记录系统的操作日志，包括用户的登录、操作记录等，以便追踪和审计系统的使用情况和异常行为。
6. 防止跨站请求伪造（CSRF）：采用合适的措施防止CSRF攻击，如使用CSRF令牌进行验证。

## 6.性能设计

1. 数据库优化：优化数据库的设计和查询，包括合理的表结构设计、索引的创建和使用、查询语句的优化等，以提高数据库的查询性能。
2. 缓存机制：使用适当的缓存机制（如Redis）来缓存常用的数据，减少数据库的访问次数，提高系统的响应速度和吞吐量。
3. 异步处理：对于一些耗时的操作，如入库单的生成和更新，可以采用异步处理方式，将耗时的操作放入消息队列或异步任务中处理，以提高系统的并发处理能力。
4. 并发控制：对于可能的并发操作，如同时多个用户对同一个入库单进行操作，需要考虑并发控制机制，如乐观锁或悲观锁，以避免数据冲突和一致性问题。
5. 前端性能优化：在前端页面中，采用合适的技术和优化策略，如使用缓存、压缩资源、减少HTTP请求等，以提高前端页面的加载速度和用户体验。

## 7.可靠性设计

1. 数据备份和恢复：定期对系统的数据进行备份，并制定合适的数据恢复策略，以防止数据丢失和系统故障。
2. 容灾和高可用性：采用容灾和高可用性的设计方案，如通过数据复制、负载均衡、故障转移等手段，确保系统在部分故障或灾难情况下仍能保持正常运行。
3. 异常处理和容错机制：对系统可能出现的异常情况进行合理的处理和容错机制设计，包括异常捕获、错误处理、事务回滚等，以保证系统的稳定性和可靠性。
4. 监控和告警：建立系统的监控和告警机制，及时发现和处理系统的异常和故障，包括对系统性能、资源利用情况、错误日志等进行监控和分析。
5. 测试和质量保证：进行全面的测试，包括单元测试、集成测试、性能测试等，以保证系统的质量和可靠性。同时，建立持续集成和自动化测试流程，确保代码的稳定性和可靠性。