

数据库系统实践课程报告

题目：jxtxzzw 物流管理系统

姓名：张臻炜

学号：10165102154

完成时间：2019 年 06 月 09 日

华东师范大学计算机科学技术系

目录

1 系统需求分析	5
1.1 系统描述	5
1.1.1 公共查询部分	5
1.1.2 后台查询部分	6
1.2 数据存储需求	6
1.3 系统常做的查询与更新	7
1.4 应用程序功能	8
1.4.1 公共查询	8
1.4.2 用户登录	8
1.4.3 单向信息更新	8
1.4.4 完整信息更新	9
2 数据库概念设计	9
2.1 确定实体和属性	9
2.2 E-R 图	10
3 数据库逻辑结构设计	15
3.1 关系模式设计	15
3.2 基本表设计	16
4 数据库物理设计和实施	20
4.1 数据库的创建	20
4.2 创建基本表	21
4.3 触发器设计	27

4.4	存储过程设计	28
5	应用程序设计	31
5.1	开发及运行环境介绍	31
5.1.1	技术栈	31
5.1.2	数据库初始方式	31
5.1.3	运行方式	32
5.2	主要功能设计	32
5.2.1	公共查询	32
5.2.2	登录	34
5.2.3	派件信息	35
5.2.4	物流跟踪	36
5.2.5	员工信息维护	36
5.2.6	包裹	39
5.2.7	仓库	40
5.3	主要界面	41
5.4	前后端通信	41
5.4.1	前端路由	41
5.4.2	拦截器	43
5.4.3	后端路由	44
5.4.4	使用 JWT 进行身份验证	45
5.5	后端其他	47
5.5.1	后端初始化	47
5.5.2	数据库初始化	48

5.5.3	异步请求	49
5.5.4	检查 Token	49
5.5.5	安全性	49
5.5.6	权限	49
5.6	用户体验优化	50
6	感想	52
6.1	已开源	52
6.2	前后端分离不太好做	52
6.3	用户体验很重要	53
6.4	JS 真的什么都能写	53
6.5	Node.js 还是太底层了	54
6.6	鄙视所有抄袭和魔改的	54
6.7	还有很多可以改进	55
	附录	55
	附录附录	55
	附录 A git 提交记录	55
	附录 B 图片索引	58
	附录 C 表格索引	59

1 系统需求分析

1.1 系统描述

物流的概念最早是在美国形成的，起源于 20 世纪 30 年代，原意为“实物分配”或“货物配送”。1963 年被引入日本，日文意思是“物的流通”。20 世纪 70 年代后，日本的“物流”一词逐渐取代了“物的流通”。中国的“物流”一词是从日文资料引进来的外来词，源于日文资料中对"Logistics"一词的翻译“物流”。

中国的物流术语标准将物流定义为：物流是物品从供应地向接收地的实体流动过程中，根据实际需要，将运输、储存、装卸搬运、包装、流通加工、配送、信息处理等功能有机结合起来实现用户要求的过程。

随着人们对物流的需求越来越大，人工乃至半自动的管理已经远远不能满足人们对物流管理中包裹跟踪、包裹派送以及员工和仓库等基础信息的管理的需求，一个功能强大的物流管理系统已经成为一个必备项。

物流管理系统应该包括公共查询部分以及后台查询部分。

1.1.1 公共查询部分

任何用户，包括员工和顾客，都可以在非登录状态下查询自己的包裹信息。

具体操作流程应该是：用户输入自己的手机号，点击查询按钮，系统调用验证码发送接口发送短信验证码¹，用户在系统中输入验证码，即可查询到所有与自己相关的包裹信息²，包括包裹的基本信息以及物流情况跟踪。

为了简化操作，本系统省略了调用短信验证码的部分³，用户输入手机号、点击查询按钮，直接显示查询到的结果。

¹用以验证用户和手机号的匹配，避免爬虫软件批量爬取所有用户手机号对应的包裹信息

²与自己相关指的是寄件人手机号或者收件人手机号之一为用户输入手机号的

³事实上也没有部署相应软件、没有拿到运营商许可，但是这不影响整个系统的使用，且未来需要加上该功能时可以非常灵活地扩展

1.1.2 后台查询部分

后台查询部分包括员工信息的录入和更新、仓库信息的录入和更新、包裹信息的录入和更新。

这部分内容仅供后台用户登录后使用,未登录状态会隐藏这部分功能。登录后的用户根据所属权限不同,只能使用各自的部分,例如人力资源相关的员工只能使用员工相关的功能而不能管理包裹信息。

1.2 数据存储需求

jtxxzzw 物流管理系统需要存储如下信息。

员工信息 有关物流系统员工的基本信息,包括工号、姓名、出生年月、权限、薪水等。

包裹信息 有关包裹的基本信息,包括包裹编号、寄件和收件双方的基本信息(姓名、电话、地址等)、运费、寄收件日期等。

仓库信息 有关物流仓库的基本信息,包括仓库位置、仓库管理员等。

用户登录信息 有关员工登录账号的信息,包括用户名、密码等。

物流跟踪信息 有关物流信息的更新,包括入库出库操作、操作人、所属仓库、操作日期等。

派件信息 某一给定包裹是由哪一位派件员派送的,用户对此次快递评分多少。

仓库管理员 该表列出了拥有仓库管理权限的员工,并为今后添加仓库管理员职称、师徒关系预留接口。

派件员 该表列出了拥有派件权限的员工,以及用户对他们的平均评分。

人力资源管理 该表列出了拥有人力资源部门相关权限的员工。

前台接待员 该表列出了拥有前台接待、包裹信息录入相关权限的员工。

运输员 该表列出了拥有运输权限的员工，并为今后添加运输员所属运输车队、运输车辆、车辆检修和保险情况预留接口。

1.3 系统常做的查询与更新

经常做的查询，或许对创建索引有影响的：

1. 包裹信息查询，包括列出所有包裹和根据用户的筛选列出指定包裹
2. 物流信息查询，包括按时间序列列出给定包裹的物流跟踪情况

根据经常做的查询，需要创建有关视图的：

无。

关于更新

1. 包裹信息录入的频率非常大，但是一般情况下录入后更新较少⁴
2. 物流跟踪信息录入的频率非常大，一旦录入后不允许修改和删除
3. 员工信息相对稳定，本系统只涉及姓名、出生年月、薪水等信息，这些信息或者一旦确定几乎永远不会被更改、或者一般用户没有修改权限，因此只需对人力资源部门开放展示和修改权限即可

⁴例如临时修改收件人地址、联系电话

1.4 应用程序功能

1.4.1 公共查询

任何用户，包括员工和顾客，都可以在非登录状态下查询自己的包裹信息。

1.4.2 用户登录

员工可以在此登录。

登录状态下访问该页面可以选择注销，或者修改密码。

预留了未来与其他系统通信的接口，例如可以访问财务管理系统导出历史 6 个月的工资和奖金。

1.4.3 单向信息更新

所谓单向信息更新，指的是快捷信息添加，且该信息只允许增加，不允许修改和删除。

派件信息 派件员可以在这个界面看到被分配给自己的、且正在派送中的信息，包括收件人地址、联系方式等，并提供一个按钮，允许派件员点击后确认送达。

物流信息 运输员可以在这个界面录入包裹的最新动态，只需给定仓库名称、包裹编号，即可在数据库中录入“包裹目前已经到达某一仓库”这一信息，系统将自动添加当前时间。

如上文所述，这些信息不具有可修改性，一旦录入就是终态，因此无需提供修改和删除功能，而之所以选择将这两个页面独立，是出于 2 方面考虑：

1. 单页面应用可以很容易地部署到移动终端、例如小型嵌入式设备等，只需要极少量的信息展示和输入（甚至只需要一个按钮）无需支持复杂的交互逻辑
2. 录入包裹编号可以很方便地与扫码枪结合，只需预先设置仓库名称，之后使用扫码枪扫描条形码录入包裹编号，即可自动提交并情况当前信息，实现快速、批量录入

1.4.4 完整信息更新

员工 员工信息的录入、修改和删除。

录入员工信息时，必须给定员工的手机号、邮箱、出生年月、薪水等。其中，手机号必须是 11 位的数字，邮箱必须满足邮箱命名规则，薪水只能包含数字 0 到 9 和小数点且必须表示一个数值。

包裹 包裹信息的录入、修改和删除。

录入包裹信息时，必须给定寄件收件双方的信息（联系方式、姓名、地址等）。其中，手机号必须是 11 位的数字，地址不得少于 10 个字，城市将提供下拉框以供选择，订单金额只能包含数字 0 到 9 和小数点且必须表示一个数值。

其中，包裹信息查询页面，将嵌入物流信息查看页面。该页面包括给定包裹目前所处的状态（已揽件、运输中、派件中、已签收），并给出按时间序列排序的物流运输信息查看。

仓库 仓库信息的录入、修改和删除。

录入仓库信息时，必须给定仓库所在地、仓库名称、仓库负责人名称。其中，城市将提供下拉框以供选择，仓库负责人将提供下拉框以供选择。

2 数据库概念设计

2.1 确定实体和属性

分析物流管理系统的管理需求，将系统中涉及到的人、包裹、仓库进行抽象，得到了系统的实体如下：

1. 派件员实体集。属性包括：派件员员工工号、派件员平均得分
2. 派件信息实体集。属性包括：包裹编号、派件员员工工号、顾客对这一次派件的评分
3. 员工实体集。属性包括：员工工号、姓名、手机号、出生日期、薪水

4. 人力资源实体集。属性包括: 员工工号
5. 城市实体集。属性包括: 城市名称、上级城市名称
6. 登录信息实体集。属性包括: 邮箱、密码、员工工号
7. 包裹接待员信息实体集。属性包括: 包裹编号、接待员员工工号
8. 包裹实体集。属性包括: 包裹编号、寄件人姓名、寄件人联系方式、寄件人地址、寄件人所在城市、寄件日期、收件人姓名、收件人联系方式、收件人地址、收件人城市、收件日期、运费、运费是否已经支付、包裹状态
9. 前台接待员实体集。属性包括: 员工工号
10. 物流信息实体集。属性包括: 出入库操作、操作日期、操作员员工工号、仓库编号、包裹编号
11. 运输员实体集。属性包括: 员工工号
12. 仓库管理员实体集。属性包括: 员工工号
13. 仓库实体集。属性包括: 仓库编号、仓库名称、仓库所在城市、仓库管理员员工工号

2.2 E-R 图

总览 总览如图 1。

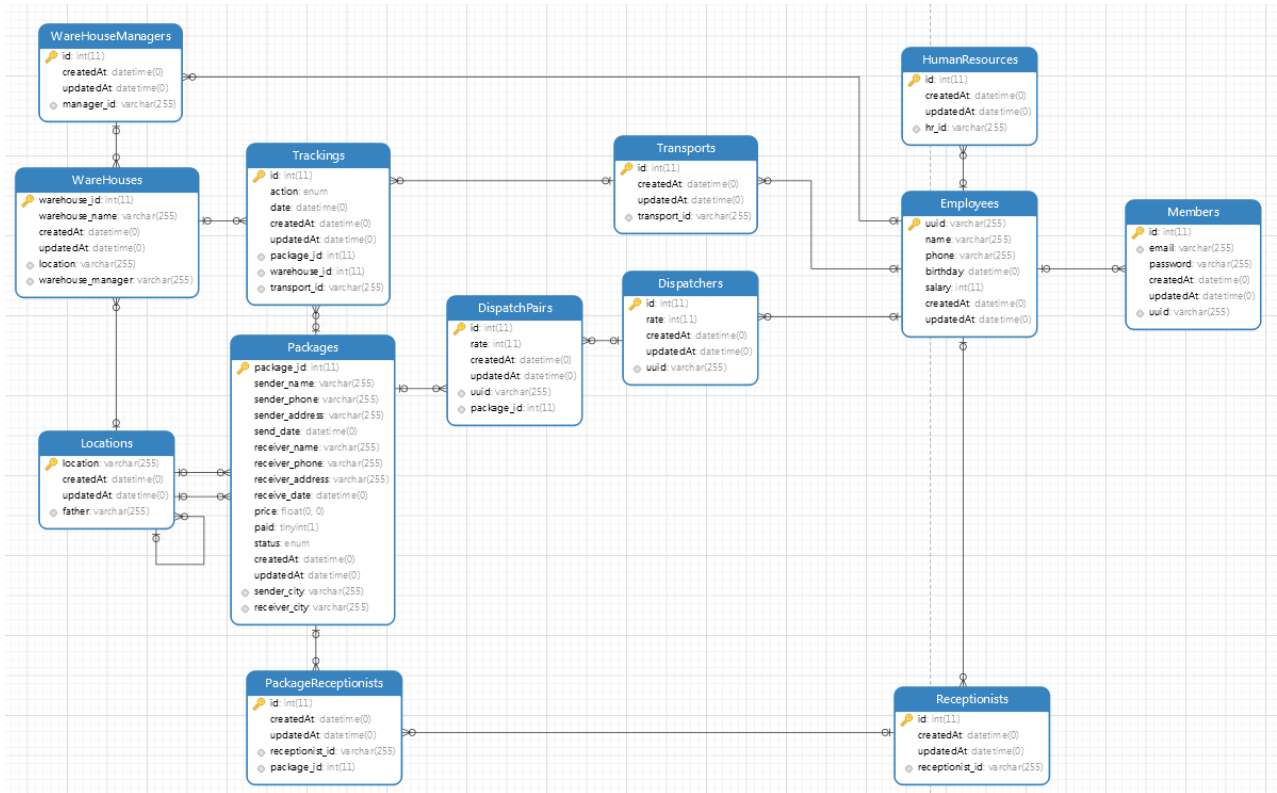


图 1: ER 图总览

员工角色相关 如图 2 表示员工及所属部门的 E-R 图，其中 **Employees** 为员工信息表，其余各表分别表示不同的职能，当员工属于某一部门时，信息将添加到对应表中，这些表采用了员工工号作为外键约束。

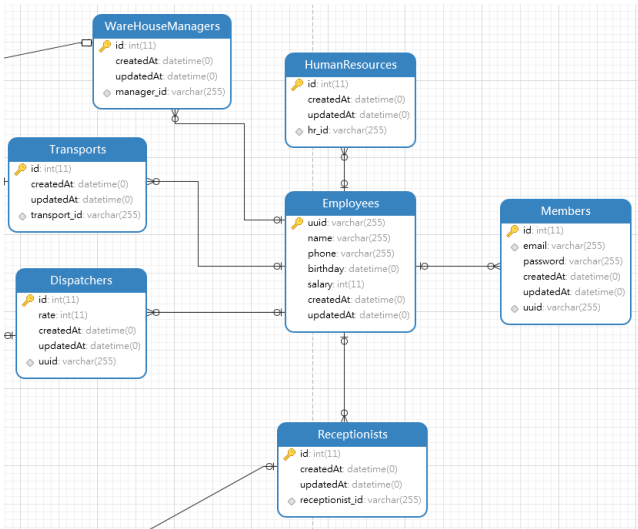


图 2: 员工角色相关 E-R 图

仓储 如图 3 表示仓库管理员及所管理仓库的 E-R 图。

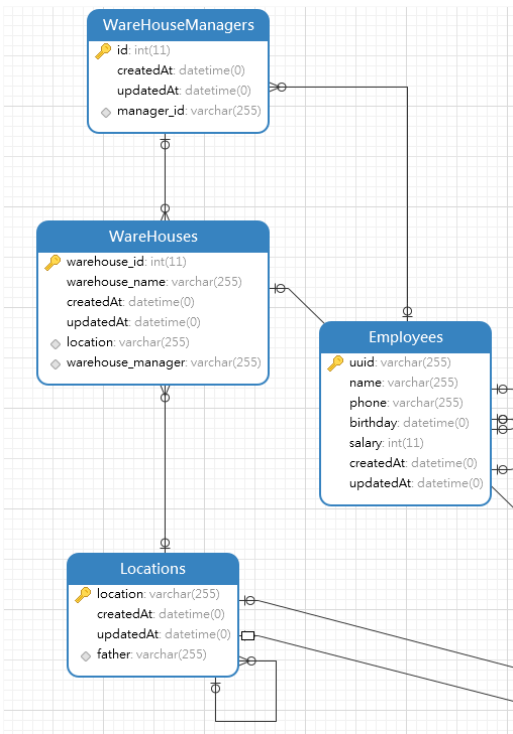


图 3: 仓储 E-R 图

包裹及物流 如图 4 表示包裹信息、寄件和收件所在城市、物流信息跟踪（按时间序列的各城市仓库周转历史）的 E-R 图。

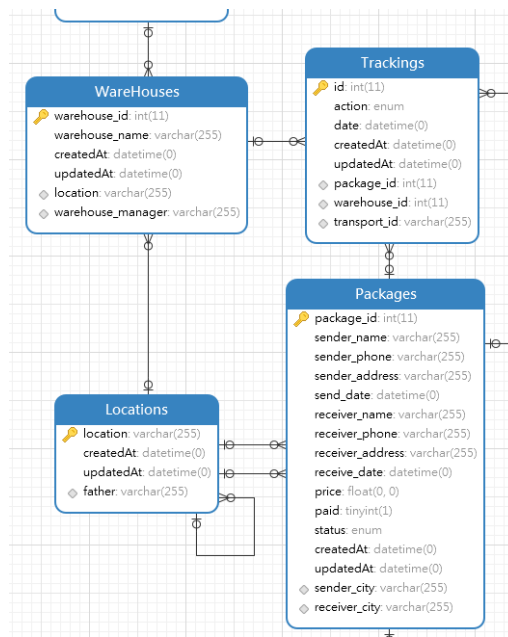


图 4: 包裹及物流 E-R 图

派件 如图 5 表示包裹派件的 E-R 图，包括包裹由哪一个派件员派送、这一次派送的用户评分、派件员基础信息及平均得分等。

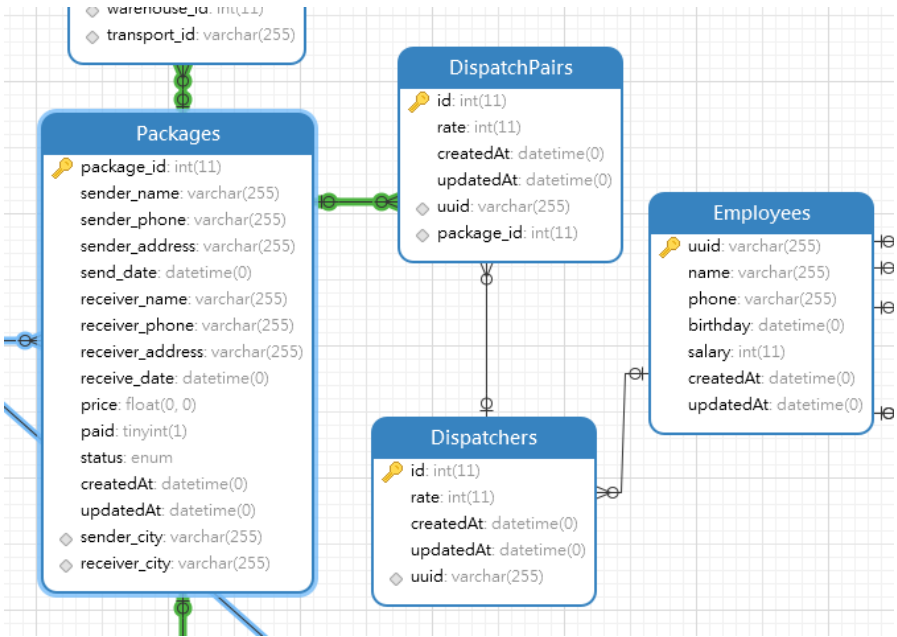


图 5：派件 E-R 图

包裹与前台接待 如图 6 表示包裹与前台接待的 E-R 图。

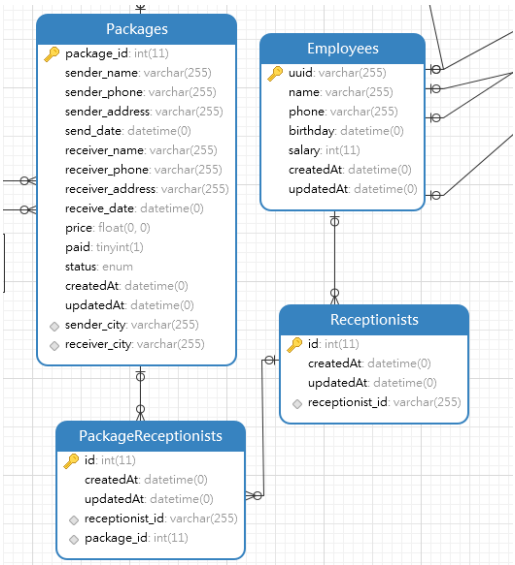


图 6：包裹与前台接待 E-R 图

3 数据库逻辑结构设计

3.1 关系模式设计

根据概念结构设计得到的 E-R 图和转换规则，得到如下关系模式⁵：

- Dispatchers 表 (自增编号、员工工号⁶、派件员平均得分、记录添加时间、记录最后一次修改时间)
- DispatchPairs 表 (自增编号、包裹编号⁷、员工工号⁸、顾客对这一次派件的评分、记录添加时间、记录最后一次修改时间)
- Employees 表 (员工工号、姓名、手机号、出生日期、薪水、记录添加时间、记录最后一次修改时间)
- HumanResources 表 (自增编号、员工工号⁹、记录添加时间、记录最后一次修改时间)
- Locations 表 (城市名称、上级城市名称¹⁰、记录添加时间、记录最后一次修改时间)
- Members 表 (自增编号、邮箱、密码、员工工号¹¹、记录添加时间、记录最后一次修改时间)
- PackageReceptionists 表 (自增编号、员工工号¹²、包裹编号¹³、记录添加时间、记录最后一次修改时间)
- Packages 表 (包裹编号、寄件人姓名、寄件人联系方式、寄件人地址、寄件人所在城市¹⁴、寄件日期、收件人姓名、收件人联系方式、收件人地址、收件人城市¹⁵、收件日期、运费、运费是否已经支付、包裹状态、记录添加时间、记录最后一次修改时间)

⁵主键用下划线标出，外键予以说明

⁶参考 Employees 表的员工工号

⁷参考 Packages 表的包裹编号

⁸参考 Employees 表的员工工号

⁹参考 Employees 表的员工工号

¹⁰参考 Locations 表的城市名称

¹¹参考 Employees 表的员工工号

¹²参考 Employees 表的员工工号

¹³参考 Packages 表的包裹编号

¹⁴参考 Locations 表的城市名称

¹⁵参考 Locations 表的城市名称

- Receptionists 表 (自增编号、员工工号¹⁶、记录添加时间、记录最后一次修改时间)
- Trackings 表 (自增编号、出入库操作、员工工号¹⁷、包裹编号¹⁸、仓库编号¹⁹、记录添加时间、记录最后一次修改时间)
- Transports 表 (自增编号、员工工号²⁰、记录添加时间、记录最后一次修改时间)
- WareHouseManagers 表 (自增编号、员工工号²¹、记录添加时间、记录最后一次修改时间)
- WareHouses 表 (仓库编号、仓库名称、仓库所在城市²²、仓库管理员²³、记录添加时间、记录最后一次修改时间)

3.2 基本表设计

基本表设计如下所示。

表 1: Dispatchers

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
rate	int(11)	否		3		派件员平均得分
uuid	varchar(255)	是			外键	派件员员工工号

¹⁶参考 Employees 表的员工工号

¹⁷参考 Employees 表的员工工号

¹⁸参考 Packages 表的包裹编号

¹⁹参考 WareHouses 表的仓库编号

²⁰参考 Employees 表的员工工号

²¹参考 Employees 表的员工工号

²²参考 Locations 表的城市名称

²³参考 WareHouseManagers 表的员工工号

表 2: DispatchPairs

属性名	数据类型	是否可空	列约束	默认值	键	解释
<code>id</code>	int(11)	否	自增、唯一	3	主键	ID
<code>rate</code>	int(11)	是				用户对当前订单的评分
<code>uuid</code>	varchar(255)	是			外键	派件员员工工号
<code>package_id</code>	int(11)	是			外键	包裹编号

表 3: Employees

属性名	数据类型	是否可空	列约束	默认值	键	解释
<code>uuid</code>	varchar(255)	否	唯一		主键	员工工号
<code>name</code>	varchar(255)	否				姓名
<code>phone</code>	varchar(255)	否				手机
<code>birthday</code>	datetime	否				生日
<code>salary</code>	int(11)	否				薪水

表 4: HumanResources

属性名	数据类型	是否可空	列约束	默认值	键	解释
<code>id</code>	int(11)	否	自增、唯一		主键	ID
<code>hr_id</code>	varchar(255)	是			外键	HR 工号

表 5: Locations

属性名	数据类型	是否可空	列约束	默认值	键	解释
<code>Locations</code>	varchar(255)	否	唯一		主键	城市
<code>father</code>	varchar(255)	是			外键	上级城市

表 6: Members

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
email	varchar(255)	否	唯一			邮箱
password	varchar(255)	是				密码
uuid	varchar(255)	是			外键	员工工号

表 7: PackageReceptionists

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
receptionist_id	varchar(255)	是			外键	接待员员工工号
package_id	int(11)	是			外键	包裹编号

表 8: Packages

属性名	数据类型	是否可空	列约束	默认值	键	解释
package_id	int(11)	否	自增、唯一		主键	包裹编号
sender_name	varchar(255)	否				寄件人姓名
sender_phone	varchar(255)	否				寄件人手机
sender_address	varchar(255)	否				寄件人地址
send_date	datetime	否				寄件日期
sender_city	varchar(255)	是			外键	寄件人城市
receiver_name	varchar(255)	否				收件人姓名
receiver_phone	varchar(255)	否				收件人手机
receiver_address	varchar(255)	否				收件人地址
receive_date	datetime	是				收件日期
receiver_city	varchar(255)	是			外键	收件人城市
price	float	否				运费
paid	tinyint(1)	否		FALSE		是否已经支付
status	ENUM	否				状态

表 9: Receptionists

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
receptionist_id	varchar(255)	是			外键	接待员工号

表 10: Trackings

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一	当前日期	主键	ID
action	ENUM	否	[' 出库', ' 入库']			操作
date	datetime	否				日期
warehouse_id	int(11)	是			外键	仓库编号
package_id	int(11)	是			外键	包裹编号
transport_id	varchar(255)	是			外键	运输员工号

表 11: Transports

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
transport_id	varchar(255)	是			外键	运输员工号

表 12: WareHouseManagers

属性名	数据类型	是否可空	列约束	默认值	键	解释
id	int(11)	否	自增、唯一		主键	ID
manager_id	varchar(255)	是			外键	仓库管理员工号

表 13: WareHouses

属性名	数据类型	是否可空	列约束	默认值	键	解释
warehouse_id	int(11)	否	自增、唯一		主键	仓库编号
warehouse_name	varchar(255)	否				仓库名称
manager_id	varchar(255)	是			外键	仓库管理员工号
location	varchar(255)	是			外键	仓库所在城市

以上各表另有 2 个字段, 表示每条记录新建的时间和最后修改的时间, 如表 14 所示。

表 14: 额外信息记录

属性名	数据类型	是否可空	列约束	默认值	键	解释
createdAt	datetime	否				记录创建时间
updateAt	datetime	否		当前时间		最后一次修改时间

4 数据库物理设计和实施

4.1 数据库的创建

使用 MariaDB 10.3.14, 数据库基本信息如下:

- 服务器类型: MariaDB
- 服务器版本: 10.3.14-MariaDB-log - Source distribution
- 协议版本: 10
- 数据库客户端版本: libmysql - mysqlnd 5.0.12-dev - 20150407
- 服务器字符集: UTF-8 Unicode (utf8)

4.2 创建基本表

基本表的创建使用 ORM 模型²⁴实现。但是这里仍将给出创建每个基本表的 SQL DDL 语句。

```
1 CREATE TABLE `Dispatchers` (  
2   `id` int(11) NOT NULL AUTO_INCREMENT,  
3   `rate` int(11) NOT NULL DEFAULT 3,  
4   `createdAt` datetime NOT NULL,  
5   `updatedAt` datetime NOT NULL,  
6   `uuid` varchar(255) DEFAULT NULL,  
7   PRIMARY KEY (`id`),  
8   KEY `uuid` (`uuid`),  
9   CONSTRAINT `Dispatchers_ibfk_1`  
10  FOREIGN KEY (`uuid`)  
11   REFERENCES `Employees` (`uuid`)  
12   ON DELETE CASCADE ON UPDATE CASCADE  
13 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `DispatchPairs` (  
2   `id` int(11) NOT NULL AUTO_INCREMENT,  
3   `rate` int(11) DEFAULT 3,  
4   `createdAt` datetime NOT NULL,  
5   `updatedAt` datetime NOT NULL,  
6   `uuid` varchar(255) DEFAULT NULL,  
7   `package_id` int(11) DEFAULT NULL,  
8   PRIMARY KEY (`id`),  
9   KEY `uuid` (`uuid`),  
10  KEY `package_id` (`package_id`),  
11  CONSTRAINT `DispatchPairs_ibfk_1`  
12  FOREIGN KEY (`uuid`)  
13   REFERENCES `Dispatchers` (`uuid`)  
14   ON DELETE SET NULL ON UPDATE CASCADE,
```

²⁴我采用的 ORM 是 Sequelize, 下同

```
15  CONSTRAINT `DispatchPairs_ibfk_2`  
16  FOREIGN KEY (`package_id`)  
17  REFERENCES `Packages` (`package_id`)  
18  ON DELETE SET NULL ON UPDATE CASCADE  
19 ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `Employees` (  
2  `uuid` varchar(255) NOT NULL,  
3  `name` varchar(255) NOT NULL,  
4  `phone` varchar(255) NOT NULL,  
5  `birthday` datetime NOT NULL,  
6  `salary` int(11) NOT NULL,  
7  `createdAt` datetime NOT NULL,  
8  `updatedAt` datetime NOT NULL,  
9  PRIMARY KEY (`uuid`)  
10 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `HumanResources` (  
2  `id` int(11) NOT NULL AUTO_INCREMENT,  
3  `createdAt` datetime NOT NULL,  
4  `updatedAt` datetime NOT NULL,  
5  `hr_id` varchar(255) DEFAULT NULL,  
6  PRIMARY KEY (`id`),  
7  KEY `hr_id` (`hr_id`),  
8  CONSTRAINT `HumanResources_ibfk_1`  
9  FOREIGN KEY (`hr_id`)  
10  REFERENCES `Employees` (`uuid`)  
11  ON DELETE CASCADE ON UPDATE CASCADE  
12 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `Locations` (  
2  `location` varchar(255) NOT NULL,  
3  `createdAt` datetime NOT NULL,
```

```
4   `updatedAt` datetime NOT NULL,  
5   `father` varchar(255) DEFAULT NULL,  
6   PRIMARY KEY (`location`),  
7   KEY `father` (`father`),  
8   CONSTRAINT `Locations_ibfk_1`  
9   FOREIGN KEY (`father`)  
10  REFERENCES `Locations` (`location`)  
11  ON UPDATE CASCADE  
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `Members` (  
2   `id` int(11) NOT NULL AUTO_INCREMENT,  
3   `email` varchar(255) NOT NULL,  
4   `password` varchar(255) DEFAULT NULL,  
5   `createdAt` datetime NOT NULL,  
6   `updatedAt` datetime NOT NULL,  
7   `uuid` varchar(255) DEFAULT NULL,  
8   PRIMARY KEY (`id`),  
9   UNIQUE KEY `email` (`email`),  
10  KEY `uuid` (`uuid`),  
11  CONSTRAINT `Members_ibfk_1`  
12  FOREIGN KEY (`uuid`)  
13  REFERENCES `Employees` (`uuid`)  
14  ON DELETE CASCADE ON UPDATE CASCADE  
15 ) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `PackageReceptionists` (  
2   `id` int(11) NOT NULL AUTO_INCREMENT,  
3   `createdAt` datetime NOT NULL,  
4   `updatedAt` datetime NOT NULL,  
5   `receptionist_id` varchar(255) DEFAULT NULL,  
6   `package_id` int(11) DEFAULT NULL,  
7   PRIMARY KEY (`id`),  
8   KEY `receptionist_id` (`receptionist_id`),
```

```
9 KEY `package_id` (`package_id`),
10 CONSTRAINT `PackageReceptionists_ibfk_1`
11 FOREIGN KEY (`receptionist_id`)
12 REFERENCES `Receptionists` (`receptionist_id`)
13 ON DELETE SET NULL ON UPDATE CASCADE,
14 CONSTRAINT `PackageReceptionists_ibfk_2`
15 FOREIGN KEY (`package_id`)
16 REFERENCES `Packages` (`package_id`)
17 ON DELETE SET NULL ON UPDATE CASCADE
18 ) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `Packages` (
2   `package_id` int(11) NOT NULL AUTO_INCREMENT,
3   `sender_name` varchar(255) NOT NULL,
4   `sender_phone` varchar(255) NOT NULL,
5   `sender_address` varchar(255) NOT NULL,
6   `send_date` datetime NOT NULL,
7   `receiver_name` varchar(255) NOT NULL,
8   `receiver_phone` varchar(255) NOT NULL,
9   `receiver_address` varchar(255) NOT NULL,
10  `receive_date` datetime DEFAULT NULL,
11  `price` float NOT NULL,
12  `paid` tinyint(1) NOT NULL DEFAULT 0,
13  `status` enum
14  (' 已揽件'0027,' 运输中',' 派件中',' 已签收')
15  NOT NULL,
16  `createdAt` datetime NOT NULL,
17  `updatedAt` datetime NOT NULL,
18  `sender_city` varchar(255) DEFAULT NULL,
19  `receiver_city` varchar(255) DEFAULT NULL,
20  PRIMARY KEY (`package_id`),
21  KEY `sender_city` (`sender_city`),
22  KEY `receiver_city` (`receiver_city`),
23  CONSTRAINT `Packages_ibfk_1` FOREIGN KEY
24  (`sender_city`) REFERENCES `Locations`
```



```
25     (`location`)
26     ON DELETE SET NULL ON UPDATE CASCADE,
27     CONSTRAINT `Packages_ibfk_2`
28     FOREIGN KEY (`receiver_city`)
29     REFERENCES `Locations` (`location`)
30     ON DELETE SET NULL ON UPDATE CASCADE
31 ) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `Receptionists` (
2   `id` int(11) NOT NULL AUTO_INCREMENT,
3   `createdAt` datetime NOT NULL,
4   `updatedAt` datetime NOT NULL,
5   `receptionist_id` varchar(255) DEFAULT NULL,
6   PRIMARY KEY (`id`),
7   KEY `receptionist_id` (`receptionist_id`),
8   CONSTRAINT `Receptionists_ibfk_1`
9   FOREIGN KEY (`receptionist_id`)
10  REFERENCES `Employees` (`uuid`)
11   ON DELETE CASCADE ON UPDATE CASCADE
12 ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4;
```

```
1 CREATE TABLE `Trackings` (
2   `id` int(11) NOT NULL AUTO_INCREMENT,
3   `action` enum(' 入库',' 出库') NOT NULL,
4   `date` datetime NOT NULL,
5   `createdAt` datetime NOT NULL,
6   `updatedAt` datetime NOT NULL,
7   `package_id` int(11) DEFAULT NULL,
8   `warehouse_id` int(11) DEFAULT NULL,
9   `transport_id` varchar(255) DEFAULT NULL,
10  PRIMARY KEY (`id`),
11  KEY `package_id` (`package_id`),
12  KEY `warehouse_id` (`warehouse_id`),
13  KEY `transport_id` (`transport_id`),
```

```
14  CONSTRAINT `Trackings_ibfk_1`
15  FOREIGN KEY (`package_id`)
16  REFERENCES `Packages` (`package_id`)
17  ON DELETE CASCADE ON UPDATE CASCADE,
18  CONSTRAINT `Trackings_ibfk_2`
19  FOREIGN KEY (`warehouse_id`)
20  REFERENCES `WareHouses` (`warehouse_id`)
21  ON DELETE SET NULL ON UPDATE CASCADE,
22  CONSTRAINT `Trackings_ibfk_3`
23  FOREIGN KEY (`transport_id`)
24  REFERENCES `Transports` (`transport_id`)
25  ON DELETE SET NULL ON UPDATE CASCADE
26 ) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `Transports` (
2    `id` int(11) NOT NULL AUTO_INCREMENT,
3    `createdAt` datetime NOT NULL,
4    `updatedAt` datetime NOT NULL,
5    `transport_id` varchar(255) DEFAULT NULL,
6    PRIMARY KEY (`id`),
7    KEY `transport_id` (`transport_id`),
8    CONSTRAINT `Transports_ibfk_1`
9    FOREIGN KEY (`transport_id`)
10     REFERENCES `Employees` (`uuid`)
11     ON DELETE CASCADE ON UPDATE CASCADE
12 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `WareHouseManagers` (
2    `id` int(11) NOT NULL AUTO_INCREMENT,
3    `createdAt` datetime NOT NULL,
4    `updatedAt` datetime NOT NULL,
5    `manager_id` varchar(255) DEFAULT NULL,
6    PRIMARY KEY (`id`),
7    KEY `manager_id` (`manager_id`),
```

```
8  CONSTRAINT `WareHouseManagers_ibfk_1`
9  FOREIGN KEY (`manager_id`)
10 REFERENCES `Employees` (`uuid`)
11  ON DELETE CASCADE ON UPDATE CASCADE
12 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
```

```
1  CREATE TABLE `WareHouses` (
2  `warehouse_id` int(11) NOT NULL AUTO_INCREMENT,
3  `warehouse_name` varchar(255) NOT NULL,
4  `createdAt` datetime NOT NULL,
5  `updatedAt` datetime NOT NULL,
6  `location` varchar(255) DEFAULT NULL,
7  `warehouse_manager` varchar(255) DEFAULT NULL,
8  PRIMARY KEY (`warehouse_id`),
9  KEY `location` (`location`),
10 KEY `warehouse_manager` (`warehouse_manager`),
11 CONSTRAINT `WareHouses_ibfk_1`
12 FOREIGN KEY (`location`)
13 REFERENCES `Locations` (`location`)
14 ON DELETE CASCADE ON UPDATE CASCADE,
15 CONSTRAINT `WareHouses_ibfk_2`
16 FOREIGN KEY (`warehouse_manager`)
17 REFERENCES `WareHouseManagers` (`manager_id`)
18 ON DELETE SET NULL ON UPDATE CASCADE
19 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
```

4.3 触发器设计

触发器设计采用 ORM 模型，这里将给出 ORM 模型定义中与触发器相关的部分。

```
1  Employees.afterCreate(async (employee) => {
2    await Members.create({
3      email: employee.uuid + '@zzw.mock.com',
```

```
4      uuid: employee.uuid,
5      password: MD5_SUFFIX. OUTER +
6          md5(MD5_SUFFIX. INNER +
7              employee.phone.toString().substr(7))
8    })
9  })
10 Employees.beforeDestroy(async (employee) => {
11    await Members.destroy({
12      where: {
13        uuid: employee.uuid
14      }
15    })
16  })
```

例如上面这段代码, 在添加新员工的时候, 自动触发将员工信息添加到登录信息表。登录信息表中的密码默认采用员工手机号后 4 位、邮箱采用员工工号加企业域名组合、员工工号采用员工信息表中的工号。

4.4 存储过程设计

本系统涉及到较多的存储过程逻辑, 这里只展示部分, 完整内容可参看源码。

存储过程设计采用 ORM 模型, 这里将给出 ORM 模型定义中与存储过程相关的部分。

当前录入的物流信息中仓库所在城市与收件人地址所在城市相同时, 自动将包裹状态更新为派件中, 并随机从派件员列表中找到一位派件员指派派送该包裹

```
1  const params = request.body
2  await WareHouses.findOne({
3    where: {
4      warehouse_name: params.warehouse_name
5    },
6    attributes: ['warehouse_id']
7  })
```

```
8 .then(async warehouse => {
9   const warehouse_id = warehouse.get('warehouse_id')
10  await Trackings.create({
11    action: params.action,
12    date: Date.now(),
13    package_id: params.package_id,
14    warehouse_id: warehouse_id,
15    transport_id: request.user.uuid
16  }).then(async () => {
17    await Packages.update({
18      status: ' 运输中'
19    }, {
20      where: {
21        package_id: params.package_id
22      }
23    })
24    // 只要有物流信息了, 就自动进入运输中这一状态
25    .then(async () => {
26      await Packages.findOne({
27        where: {
28          package_id: params.package_id
29        },
30        attributes: ['receiver_city']
31      })
32      .then(async city1Project => {
33        const city1 = city1Project.get('receiver_city')
34        await WareHouses.findOne({
35          where: {
36            warehouse_id: warehouse_id
37          },
38          attributes: ['location']
39        })
40        .then(async city2Project => {
41          const city2 = city2Project.get('location')
42          if (city1 === city2) {
43            await Packages.update({
44              status: ' 派件中'
```

```
45     }, {
46         where: {
47             package_id: params.package_id
48         }
49     })
50     // 如果当前城市和收件人城市相同, 则进入派件状态
51     const Dispatchers =
52         orm.import('../database/models/Dispatchers')
53     await Dispatchers.count()
54     .then(async count => {
55         const rand = Math.floor(Math.random() * count)
56         await Dispatchers.findOne({
57             offset: rand
58         })
59         .then(async randomProject => {
60             // 随机一个派件员
61             const randomUUID = randomProject.get('uuid')
62             const DispatchPairs =
63                 orm.import('../database/models/DispatchPairs')
64             await DispatchPairs.create({
65                 package_id: params.package_id,
66                 uuid: randomUUID
67             })
68             // 加入派件信息表
69             .then(() => {
70                 response.sendStatus(200)
71             })
72         })
73     })
74     } else {
75         response.sendStatus(200)
76     }
77 })
78 })
79 })
80 }, () => {
81     response.sendStatus(403)
```

82 })

83 })

5 应用程序设计

5.1 开发及运行环境介绍

5.1.1 技术栈

前端

1. Node.js
2. Vue.js
3. Electron.js
4. iView

后端

1. Node.js
2. Express.js
3. Sequelize

5.1.2 数据库初始方式

有 3 种方式进行数据库初始化：

1. 直接运行程序，使用我的服务器上的数据库进行演示
2. 修改 `/server/database/util.js` 的配置信息为你自己的数据库地址、用户名、密码，然后导入我提供的 sql 数据

3. 修改 `/server/database/util.js` 的配置信息为你自己的数据库地址、用户名、密码，运行 `npm run init-database` 命令，系统将自动初始化数据库，并新建初始超级管理员

5.1.3 运行方式

首先需要安装 `Node.js` 和 `npm`。

然后下载源码，在源码根目录下运行 `npm install`，将自动下载并安装依赖。

最后运行 `npm run run` 启动程序。也可以通过分别运行 `npm run server` 和 `npm run dev` 启动后端和前端。

5.2 主要功能设计

这里展现各个功能的演示画面和部分代码，完整代码过于冗长不便展示，可以通过阅读源码的方式了解全部逻辑。

5.2.1 公共查询

该页面提供一个输入框，允许用户输入手机号进行查询。

点击查询按钮之后，系统将从数据库找出寄件人联系方式或收件人联系方式为该手机号的所有包裹，并罗列包裹的基本信息、包裹所处状态、物流跟踪信息等。

13012345678

🔍 查询

包裹ID	寄件人姓名	寄件人联系电话	寄件人所在城市	寄件人地址	寄件时间	收件人姓名	收件人联系电话	收件人所在城市	收件人地址	收件时间	物流信息	状态	操作
<div>输入包ID</div>	<div>输入寄件人姓名</div>	<div>输入寄件人电话</div>	<div>输入寄件人城市</div>	<div>输入寄件人地址</div>		<div>输入收件人姓名</div>	<div>输入收件人电话</div>	<div>输入收件人城市</div>	<div>输入收件人地址</div>		<div>输入物流ID</div>	<div>请选择</div>	
1	张三	13012345678	上海/宝山	上海宝山123456789	2019-06-07T16:00:00.000Z	李四	13112345678	浙江/杭州	浙江杭州12345678		>	已签收	<div>评价</div>

共 1 条

<1>

10 条/页

图 7: 公共查询

对于已签收的包裹还会给出一个评价按钮，用户点击该按钮后，可对这一次配送进行评分，评分提交后会保存进数据库，并自动计算和更新该派件员的平均得分。

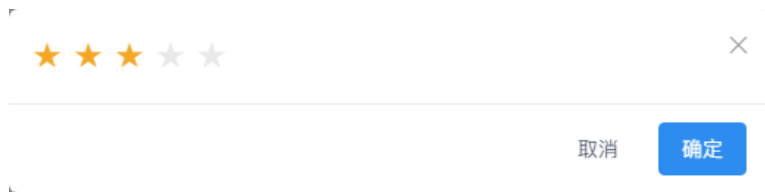


图 8: 用户评分

部分代码如下:

```
1 <Card>
2   <filter-table @on-search="onSearch"
3     :data="rowData"
4     :columns="tableColumns" />
5   <div style="margin: 10px; overflow: hidden">
6     <div style="float: right">
7       <Page :total="total"
8         :current="1"
9         @on-change="changePage"
10        show-total
11        show-sizer
12        :page-size="pageSize"
13        :page-size-opts="sizer"
14        @on-page-size-change="changePageSize"/>
15     </div>
16   </div>
17 </Card>
18 <Modal
19   v-model="toRate"
20   @on-ok="rateOnOK"
21   @on-cancel="rateOnCancel">
22   <Rate v-model="package_rate" />
23 </Modal>
```

5.2.2 登录

该页面根据用户是否登录有不同的展示。

当用户未登录时，页面提供用户名输入框和密码输入框，用户输入后点击登录按钮，系统将请求发送到后端，后端返回登录成功或失败。


A login form with two input fields: '用户名' (Username) and '密码' (Password). Below the fields is a blue button labeled '登录' (Login).

图 9：用户登录

当登录失败时，弹出消息框，说明用户名或密码错误，用户可关闭消息框后再次登录，若登录成功，将展示个人信息页面，包括修改密码等功能。



图 10：用户登录失败

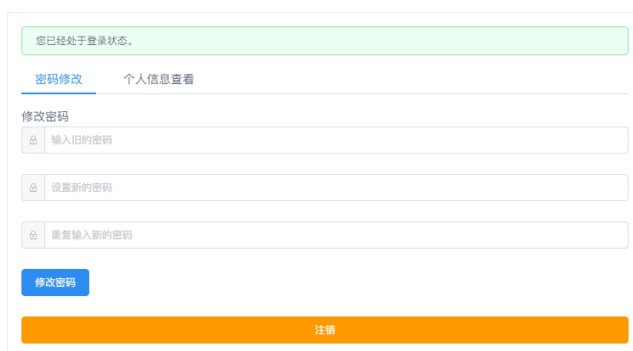
A user profile page. At the top, it says '您已经处于登录状态。' (You are already logged in). Below this are two tabs: '密码修改' (Change Password) and '个人信息查看' (View Personal Information). The '密码修改' tab is active. It contains three input fields: '输入旧的密码' (Enter old password), '设置新的密码' (Set new password), and '重复输入新的密码' (Repeat new password). Below these fields is a blue button labeled '修改密码' (Change Password). At the bottom of the page is an orange button labeled '注销' (Logout).

图 11：用户登录成功

5.2.3 派件信息

该页面仅在员工登录后可见。

系统会从数据库搜索出被分配给本人派送的且还在派送状态的所有包裹，并以表格的形式展示，展示的数据隐藏了无关信息，只保留寄件收件双方的必要联系信息。

在表格的每一行还提供一个确认签收的按钮，派件员点击该按钮后，会将该包裹标记为已送达，表格中不再显示这条记录。

只能查看分配给自己的、且正在派件中的订单。

包裹ID	寄件人姓名	寄件人联系电话	收件人姓名	收件人联系电话	收件人所在城市	收件人地址	操作
<input type="text" value="输入包裹ID"/>	<input type="text" value="输入寄件人姓名"/>	<input type="text" value="输入寄件人电话"/>	<input type="text" value="输入收件人姓名"/>	<input type="text" value="输入收件人电话"/>	<input type="text" value="输入收件人城市"/>	<input type="text" value="输入收件人地址"/>	
4	王六	18945612343	六三	18948936426	北京	北京内环中路南海小区3楼201室	标记为已签收

共 1 条 < 1 > 10 条/页

图 12: 派件信息查看

部分代码：

```
1 {
2   title: ' 操作',
3   key: 'action',
4   render: (h, params) => {
5     const _this = this
6     return h('div', [
7       h('Button', {
8         props: {
9           type: 'warning',
10          disabled: this.lackingAuth
11        },
12        style: {
13          margin: '5px 5px 5px 5px'
14        },
15        on: {
16          click: async function () {
```

```
17         await _this.dispatchDone(params.row.package_id)
18     }
19 }
20 }, ' 标记为已签收')
21 ]})
22 }
23 }
```

5.2.4 物流跟踪

该页面仅在员工登录后可见。

该页面提供 2 个下拉框和 1 个输入框，分别需要选择出入库操作和所在仓库名称，选择完成后输入包裹编号并提交，系统会将对该包裹的这一次操作保存到数据库中，并绑定当前操作时间。



图 13: 运输状态录入

5.2.5 员工信息维护

该页面仅登录后可见。

用户登录后将首先看到以表格的形式展示的所有员工的信息。

表格项目过多时，提供分页功能，表格每一行提供修改和删除按钮，分别进行修改和删除操作，表格顶部提供新增按钮点击可添加新员工的信息，点击新增或修改按钮后将进入员工信息的维护页面。

员工工号	姓名	手机号	出生年月	月薪	操作
<input type="text" value="输入员工工号"/>	<input type="text" value="输入姓名"/>	<input type="text" value="输入手机号"/>	<input type="text" value="输入出生年月"/>	<input type="text" value="输入月薪"/>	<input type="button" value="新建"/>
02e933c3-db91-42f6-9ba4-500eec2f5447	李劲松	12235585566	1971-05-07	5000	<input type="button" value="修改"/> <input type="button" value="删除"/>
0e9beb4b-8f1f-4f74-8399-d9b81190c087	WH	11111111111	2019-06-07	1111	<input type="button" value="修改"/> <input type="button" value="删除"/>
1fd5241c-3d7d-4aa3-ad0e-591c9f430279	DP	11111111111	2019-06-07	1111	<input type="button" value="修改"/> <input type="button" value="删除"/>
3d6e9683-32bb-4ae9-96c8-7320a9462d81	杜晨百	15855222101	1997-02-04	5000	<input type="button" value="修改"/> <input type="button" value="删除"/>
4f5611e2-e5ab-4e96-8ec4-f5fc0231943d	HR	11111111111	2019-06-07	1111	<input type="button" value="修改"/> <input type="button" value="删除"/>
5820b714-fa65-4ceb-b497-1205e0e49521	晨李晓慧	13654897564	1986-03-20	1000	<input type="button" value="修改"/> <input type="button" value="删除"/>
67456525-a0a6-4b3c-ba71-f67908505250	蒋谷军	18412556633	1998-05-04	3350	<input type="button" value="修改"/> <input type="button" value="删除"/>
7966cd50-ee8e-46ac-bdeb-57086a5ab65f	李世凯	15566633252	1989-12-09	12000	<input type="button" value="修改"/> <input type="button" value="删除"/>

图 14: 员工信息

维护页面包括一个表单，填写表单中各个项目后点击提交。

该页面在新增员工时，表单各个项目为空。在修改员工信息时，表单各个项目的初始值为数据库中该员工的已有信息。

用户输入时和点击提交按钮后都会触发校验器，校验输入的信息是否正确。

员工工号02e933c3-db91-42f6-9ba4-500eec2f5447

* 姓名李劲松

电话12235585566

* 邮箱12235585566@163.com

* 出生年月1971-05-07

* 权限

☐ 人力资源权限

☒ 前台接待权限

☐ 运输权限

☐ 派件权限

☒ 仓储权限

月薪5000

修改

重置密码

图 15: 员工信息维护

特别的，在修改员工信息的页面，同时提供了一个重置密码的选项，点击该按钮后，后端会随机生成一个密码，并重置该员工的密码。重置成功后，新密码会以邮件的形式发送到员工的邮箱。除本人外不再有其他知道密码，是出于安全考虑²⁵。

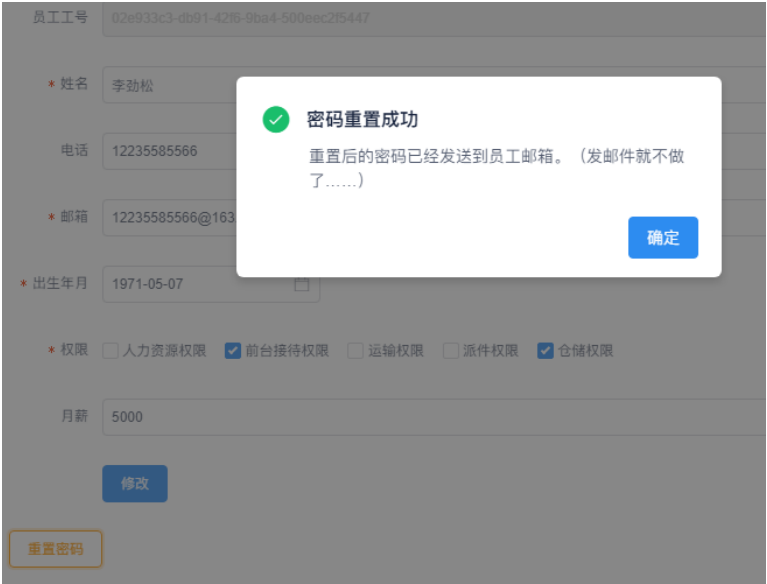


图 16: 重置密码

这些页面同时支持搜索。

收件人姓名	收件人联系电话	收件人所在城市	收件人地址	收件时间	物流信息	状态	操作
<input type="text" value="输入收件人姓名"/>	<input type="text" value="输入收件人电话"/>	<input type="text" value="输入收件人城市"/>	<input type="text" value="输入收件人地址"/>		<input type="text" value="输入物流信息"/>	派件中	新建
六三	18948936426	北京	北京内环中路南海小区3幢201室		>	派件中	修改 删除
张发	15689632156	["上海", "虹口"]	上海虹口区四海宁路文三广场		>	派件中	修改 删除
跳跳	13659486465	["上海", "宝山"]	上海宝山共康西路二小区		>	派件中	修改 删除

图 17: 搜索

部分代码：

```
1 this.trim(this.formItem)
```

²⁵避免出现这种情况：管理员后台修改某员工密码，然后使用修改后的密码登录员工账号滥权。因此，不直接提供修改密码的功能，只能重置成随机密码。

```

2  const _this = this
3  await this.$http.post('http://127.0.0.1:3000/Employee/Add', this.formItem)
4    .then(() => {
5      _this.$Message.success(' 操作成功! ')
6      router.push('/EmployeeManage')
7    })
8    .catch((error) => {
9      _this.$Modal.error({
10        title: ' 操作失败',
11        content: error.data
12      })
13    })

```

5.2.6 包裹

该页面仅登录后可见。

该页面与员工页面类似。

包裹ID	寄件人姓名	寄件人联系电话	寄件人所在城市	寄件人地址	寄件时间	收件人姓名	收件人联系电话	收件人所在城市	收件人地址	收件时间	物流信息	状态	操作
<input type="text" value="输入包ID"/>	<input type="text" value="输入寄件人姓名"/>	<input type="text" value="输入寄件人电话"/>	<input type="text" value="输入寄件人城市"/>	<input type="text" value="输入寄件人地址"/>		<input type="text" value="输入收件人姓名"/>	<input type="text" value="输入收件人电话"/>	<input type="text" value="输入收件人城市"/>	<input type="text" value="输入收件人地址"/>		<input type="text" value="输入物流信息"/>	<input type="text" value="选择状态"/>	<button>新建</button>
1	张三	13012345678	上海/宝山	上海宝山1234 56789	2019-06-07	李四	13112345678	浙江/杭州	浙江杭州1234 5678		>	已收件	<button>修改</button> <button>删除</button>
2	王五	13505975684	["北京"]	北京四季青三 楼7号房间	2019-05-26T1 6:00:00.000Z	赵三	13965497857	["浙江"]	浙江乌镇北栅 桥头村11组3号		>	运输中	<button>修改</button> <button>删除</button>
3	张桥	16587945893	["浙江","杭州"]	浙江杭州文三 路北斗小区	2018-05-03T1 6:00:00.000Z	周记	14879586821	["上海","普陀","环球港"]	上海普陀区环 球港三楼	2019-06-08T1 5:16:20.000Z	√	已签收	<button>修改</button> <button>删除</button>

图 18: 包裹页面

同时给出物流跟踪信息查看，给定一个包裹编号，将会展示包裹的物流变化情况。

展示内容包括 2 部分，第 1 部分是包裹所属状态，状态以进度条的方式展示，第 2 部分展示包裹于何时到达何地，以时间轴的方式展示。



图 19: 物流跟踪

5.2.7 仓库

该页面仅登录后可见。

该页面与员工页面类似。

5.3 主要界面



图 20: 主要界面

5.4 前后端通信

我的系统采用了前后端分离的方法，因此，需要有一个方法能够在前端和后端各自判断用户身份，以及做好权限跳转。

5.4.1 前端路由

前端路由采用 `vue-router` 和 `axios`，前端数据存放采用 `vuex`。

当用户访问的页面需要登录后可见，那么会发送一个请求给后端，请求包括“跳转自”页面和“跳转到”页面，即 `from` 和 `to` 2 个信息，后端根据用户判断是否有权限访问对应页面²⁶，并返回结果。

²⁶后端身份验证在后文解说

```
1 // 每一个路由之前都做一次检验, 除非是登录页面或者公共页面, 其他页面都要触发一个权限检查
2 router.beforeEach(async (to, from, next) => {
3   if (to.name !== 'Login' && to.name !== 'Welcome'
4     && to.name !== 'UserQuery' && to.name !== 'Tracking') {
5     Vue.http.post('http://127.0.0.1:3000/User/Auth', {
6       from: from.name,
7       to: to.name
8     })
9     .then(
10      response => {
11        if (response.data.token) {
12          this.$store.dispatch('login', response.data.token)
13        }
14        return response
15      },
16      error => {
17        const errRes = error.response
18        if (errRes.status === 401) {
19          this.$store.dispatch('logout')
20          router.push('/Login')
21        } else if (errRes.status === 403) {
22          // 跳转到没有权限的页面
23        }
24        return Promise.reject(error.message)
25        // 返回接口返回的错误信息
26      }
27    )
28    .catch(e => {
29      console.log(e)
30    })
31  }
32  next() // 通过, 则继续到下一层路由中间件
33 })
```

当用户登录成功时, 会自动将 Token 信息保存, 当用户点击注销按钮后, 或前端路由收

到 401 响应时，会清空 Token 信息。

```
1  if (project == null) {
2    response.json({
3      success: false,
4      token: ''
5    })
6  } else {
7    const token = jwt.sign(
8      project.get(),
9      secretKey,
10     {
11       expiresIn: 86400
12     }
13   )
14   response.json({
15     success: true,
16     token: token
17   })
18 }
```

5.4.2 拦截器

请求拦截器 对于每一个发往后端的请求，都拦截下来，强制添加一个 Authorization 头部。包括用户登录成功后保存的 Bearer Token，如果用户没有登录，则该字段为空。

```
1  // axios 拦截器
2  // 拦截请求，给所有的请求都带上 token
3  axiosInterceptor.interceptors.request.use(request => {
4    const token = store.state.Auth.token
5    // const JWTToken = window.localStorage.getItem('jxtæzzw_jwt_token')
6    const JWTToken = token
7    if (JWTToken) {
8      request.headers['Authorization'] = `Bearer ${JWTToken}`
9    }
10   })
```

```
9   }  
10  return request  
11  })
```

响应拦截器 对于后端发往前端的所有响应，全部拦截下来，拦截器会检查状态是否成功。对于成功的状态（200），拦截器放行；对于未授权的状态（401），拦截器跳转到登录页面；对于已授权但权限不够的状态（403），拦截器弹出错误信息；对于其他状态（406 或者 50X），拦截器抛出异常。

```
1  // 拦截响应，遇到 token 不合法则报错  
2  axiosInterceptor.interceptors.response.use(  
3    response => {  
4      if (response.data.token) {  
5        store.dispatch('login', response.data.token)  
6      } else {  
7      }  
8      return response  
9    },  
10   error => {  
11     const errRes = error.response  
12     if (errRes.status === 401) {  
13       store.dispatch('logout')  
14       router.push('/Login')  
15     } else {  
16       // 其他错误信息处理  
17     }  
18     return Promise.reject(error.message)  
19     // 返回接口返回的错误信息  
20   }  
21 )
```

5.4.3 后端路由

使用 Express Router 进行路由中间件处理。

后端路由的工作流程是，收到请求时，请求会逐层、依次通过每一层中间件，当前层按照当前的逻辑处理完成后，将请求和当前层处理的结果一传递给下一层。

```
1 // 注册路由
2 const express = require('express')
3 const router = express.Router()
4
5 // router.use((request, response, next) => {
6 //   // 任何路由信息都会执行这里面的语句
7 //   console.log('this is a api request!')
8 //   // 把它交给下一个中间件，注意中间件的注册顺序是按序执行
9 //   next()
10 // })
```

5.4.4 使用 JWT 进行身份验证

JWT 定义了加盐字符串以及无需经过校验的公共 API。

```
1 const expressJwt = require('express-jwt')
2 const {secretKey} = require('./salt')
3
4 const jwtAuth = expressJwt({
5   secret: secretKey,
6   credentialsRequired: true // 设置为 false 就不进行校验了，游客也可以访问
7 })
8
9 .unless({
10   path: [
11     '/User/Login',
12     '/Package/Tracking',
13     '/UserQuery/Query',
14     '/UserQuery/Count',
15     '/DispatchPair/ChangeRate',
16     '/Dispatcher/Rate',
17     '/Dispatcher/Info'
```

```
17     ]
18   })
19
20 module.exports = jwtAuth
```

所有请求过来都会进行身份验证, 只有通过了 JWT 校验的请求才会继续发往下一层, 否则, JWT 中间件会返回 401。

```
1 // 注册路由
2 const jwtAuth = require('./jwt')
3 router.use(jwtAuth)
```

通过 JWT 校验之后, 依次通过其他中间件, 最后进行错误处理。

```
1 // 处理 404
2 router.use((request, response, next) => {
3   let err = new Error('Not Found')
4   err.status = 404
5   next(err)
6 })
7
8 // 处理 5 错误
9 router.use((err, request, response, next) => {
10   if (err.name === 'UnauthorizedError') {
11     response.status(401).json({
12       message: 'invalid token',
13       error: err
14     })
15   } else {
16     response.status(err.status || 500)
17     response.json({
18       message: err.message,
19       error: err
```

```
20     })
21   }
22 })
```

5.5 后端其他

5.5.1 后端初始化

后端初始化过程首先定义了与跨域相关的头部，随后注册了后端路由。

```
1  const express = require('express')
2  const app = express()
3  const session = require('express-session')
4  const bodyParser = require('body-parser')
5
6  const router = require('./router')
7  app.use(function (req, res, next) {
8    res.header('Access-Control-Allow-Origin', 'http://localhost:9080')
9    res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,PATCH,OPTIONS')
10   res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, A
11   res.header('Access-Control-Allow-Credentials', 'true')
12   if (req.method === 'OPTIONS') {
13     res.end()
14   } else {
15     next()
16   }
17 })
18 app.use(bodyParser.json())
19 app.use(bodyParser.urlencoded({extended: false}))
20 app.use('/', router)
21 app.set('trust proxy', 1)
22 app.use(session({
23   secret: 'jxtxzzw_secretKey_b6vj$tV<yw2T!4dD',
24   resave: false,
25   saveUninitialized: true
```

```
26  )))
27
28  app.listen(3000)
29  console.log('success listen at port: 3000.....')
```

5.5.2 数据库初始化

数据库的初始化，提供了一个初始化脚本，按照数据表的依赖关系，强制删除和重建数据表，并录入初始的城市信息和超级管理员帐号，初始化时会授予超级管理员所有权限。

```
1  const MODELS = []
2
3  for (const x of MODEL_INIT_LIST) {
4    MODELS.push(orm.import('./models/' + x))
5  }
6
7  async function forceSyncModels(MODELS) {
8    for (const model of MODELS.reverse()) {
9      await model.sync({force: true})
10   }
11 }
12
13 async function forceDropModels(MODELS) {
14   for (const model of MODELS.reverse()) {
15     await model.drop()
16   }
17 }
18
19 orm
20   .authenticate()
21   .then(async () => {
22     console.log('Connection has been established successfully.')
23     await forceDropModels(MODELS)
24     await forceSyncModels(MODELS) // 坑: reverse 是作用在原数组上的，不是搞了个新的返回
25     console.log('Sync() succeed.')
```



```
26     console.log(' 导入默认城市数据……')
27 //……
28     console.log(' 新建超级管理员账户……')
29 //……
30     console.log('=====')
31     console.log(' 初始化完成，请不要再次执行该函数，否则会清空所有数据并重新建表！')
32     console.log(' 超级管理员：admin@jxtxzzw.com，密码：jxtxzzw')
33     console.log(' 超级管理员具有所有权限，一旦完成初始信息录入，建议立即废止账户！')
34     process.exit()
```

5.5.3 异步请求

所有异步相关的代码都加了 `async` 和 `await`。

5.5.4 检查 Token

对于给定的 API，如果需要用户授权，则使用 JWT 解码，然后取出 UUID，并判断 Token 是否过期。

```
1 router.post('/Employee/Privilege', jwt_decode({
2   secret: secretKey
3 }), async (request, response) => {
```

5.5.5 安全性

密码用了 MD5 加密，加密前和加密后都添加了前缀用以混淆。

```
1 MD5_SUFFIX. OUTER + md5(MD5_SUFFIX. INNER + this.loginForm.password)
```

5.5.6 权限

API 接口与数据库交互前都做了权限判断。

5.6 用户体验优化

我做了大量的优化用户体验的工作，以下仅列出部分：

1. 成功就不要弹出 Modal 让用户再点一次，只要 Message 就好，失败必须弹出 Modal 手动确认
2. 表单必须在用户输入的同时就 oninput validate，不要发到后台校验失败了再弹错误
3. 有校验没有通过的表单就不给提交
4. 表单提交失败不能清空表单
5. 当页面上某一元素变化²⁷，不刷新整个页面²⁸，只要局部刷新
6. 401 错误直接重定向到登录页面
7. 新增一条记录成功之后返回前一页
8. 点击遮罩层可以取消，点击 ESC 可以取消，不必一定点取消按钮或者右上角叉叉
9. 下拉框可以搜索，例如城市页面或者仓库选择页面，用户可以直接输入关键字进行全文搜索²⁹
10. 查询订单不用新开一个页面，直接当前页面批量 Render，且支持收起和展开
11. 最后一个输入框³⁰中按下回车，等效于触发 Primary Button

除此而外还做了很多优化用户体验的工作，细说之下可能又是一篇报告，这里仅列出 `inject reload` 代替全局刷新的部分代码。

Vue 中刷新页面的方法有很多，但是用户体验都不太好，如下：

1. `this.$router.go(0)`，页面会一瞬间的白屏，体验不是很好

²⁷例如状态更新、例如删除了一行记录

²⁸刷新整个页面最简单，但是用户体验极差

²⁹例如用户可以输入“华东”进行全文搜索，下拉框将自动搜索所有满足“华东”的地址，点击后一键录入完整城市列表，而不必在很长很长的下拉列表中依次选择“上海” - “普陀”……

³⁰例如登录页面的用户名和密码输入框，密码输入框就是最后一个输入框

2. 用 vue-router 重新路由到当前页面，是没有用的

3. `location.reload()` 也是一样，画面一闪，体验不是很好

局部刷新可以用 `provide / inject` 组合。原理是允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。在 `App.vue` 声明 `reload` 方法，控制 `router-view` 的显示或隐藏，从而控制页面的再次加载。

```
1 provide () {
2   return {
3     reload: this.reload
4   }
5 },
6 data () {
7   return {
8     isRouterAlive: true
9   }
10 },
11 methods: {
12   reload () {
13     this.isRouterAlive = false
14     this.$nextTick(function () {
15       this.isRouterAlive = true
16     })
17   }
18 }
```

在需要用到刷新的页面。在页面注入 `App.vue` 组件提供的 `reload` 依赖，在逻辑完成之后直接 `this.reload()` 调用，即可刷新当前页面。

6 感想

6.1 已开源

终于把这个大作业给做完了，花了很多心思，边边角角都打磨过了，可以说考虑很周全了。

2019 年 04 月 05 日至 2019 年 06 月 09 日³¹，累积 104 次 `commit`，如图 21。

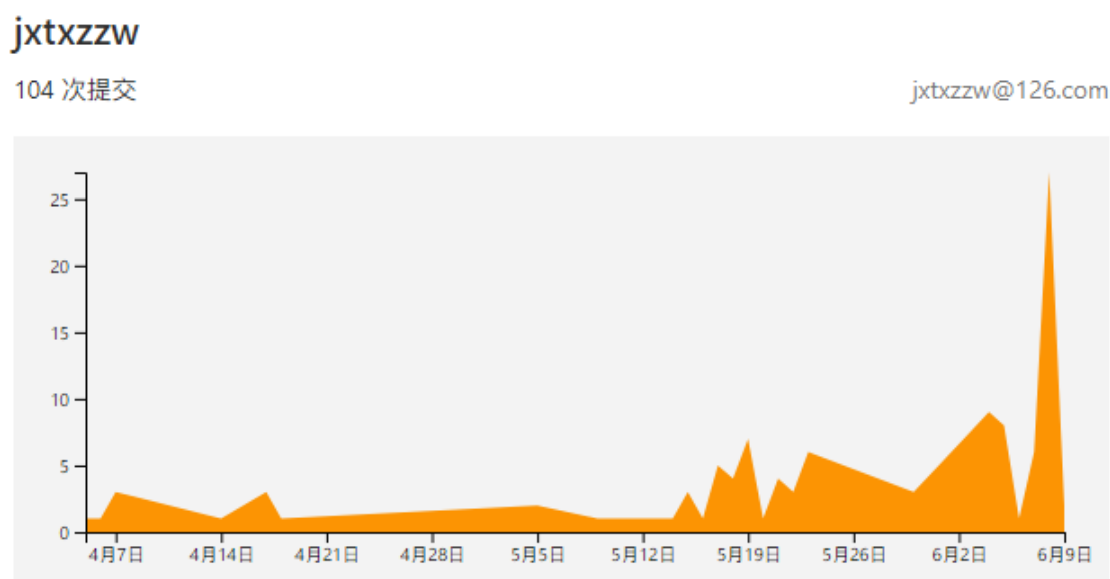


图 21: Git 提交记录

已开源在我的 GitLab 上: <https://gitlab.jtxzzw.com/jtxzzw/moneydb>

6.2 前后端分离不太好做

前后端分离是真的累。很多东西开 API 的时候，因为觉得现有 API 不能满足需求、但是开了以后又觉得，这么点小事又要开一个新的 API，以后怎么维护呀。

不像前后端一起开发的时候，数据发来，直接判断，然后根据 `if ... else ...` 决定输出什么东西到页面，就像 Python + Django 或者 PHP + HTML 或者 JSP + Servlet 一样，后端

³¹其实真正的开发周期可能是 5 月 26 日至 6 月 9 日，前面的时间都是做做玩玩、玩玩做做……

可以当前端用，直接输出 HTML 代码就行了。例如，请求中只包含员工工号，目前我想要在某张表中查信息然后输出，过两天我想要查电子邮件，我就顺手后端再去查一次数据库，然后把结果输出到 HTML 文件中对应的位置就好了。而前后端分离的话，这只能再开一个 API 了。

我彻底分离以后，后端的東西必須通過接口給前端，用 json 的格式，前端再做渲染。就很麻煩，很多事情本來可以很簡單直接輸出的，現在都要做很多事。

不過前後端分離是大勢所趨、是好事。

6.3 用戶體驗很重要

身邊一些同學做數據庫大作業的時候抱着這樣一種心態：「局部刷新太复杂了，我就刷新整个页面吧」、「这个功能做这么好，老师反正也不知道，不做了」、「就随便写一个 if else 判断权限糊弄一下老师吧」、「用户体验是什么，没有的，不存在的」、「我就只支持这些操作，他就必须这样做」、「下拉框还能搜索？表单在用户输入过程中实时 validate?」、「跳回去？自动保存？不做不做」……

没想到各种能力层次的学生都有抱着如此心态的人，很是感慨万千。

我想引用 Fenng 大文章中的一句话，怎样算是「用户体验」良好？就是你做的事情必须为对方考虑，而且，你的确考虑到了。有的时候，我们把为对方考虑这种行为，也叫做同理心。

6.4 JS 真的什么都能写

以前一直有句话，*Java* 是后端，*JavaScript* 是前端，甚至有传闻说我国某校某学生用 JS 写的前后端毕业设计，被答辩老师质疑，说 JS 怎么可能写后端，你这作品只有前端，只能拿一半分。

不过现在，真的 *JavaScript* 已经前后端通吃了——似乎离全栈又进了一步呢。（大误

6.5 Node.js 还是太底层了

还是太底层了，很多东西都没有，需要自己一个个做，尽管有很多现成的插件，但是开发起来还是非常累。

Django 的话，很多都做了——登录什么的，不仅仅是登录的后端逻辑有现成的，可以直接调用，甚至前端页面都写好了，几个文本框、几个按钮，还有注册、登录、找回密码的页面……

据我所知，DJango 连用户登录的数据库、权限数据库都默认建好了，密码的加密传输也不需要考虑，一键直接调用检查表单数据合法性，然后发到后端。权限的话，DJango 开了一张 User-Permission 的表，记录了每位用户对哪些表有操作权限，并将表模型一一映射成 ID。

Django 甚至自带授权验证和跨域请求，csrftoken，揉在一起了，根本不用了解原理，直接发过去，怎么加密的、怎么解密的、怎么验证的，都不用管了，开发者只要知道这么用，剩下的，框架都帮做掉了。

我是分开一个个自己实现的。对比使用 Django 同学的开发效率，好羡慕啊，简直就没我们什么事了，开发效率是高啊！看来我有空可以学习 Python + Django 了。

不过，我也趁此机会了解了很多偏向底层的東西，Render 函数啊、拦截器啊、中间件啊，都了解了，是好事。

6.6 鄙视所有抄袭和魔改的

每年谈及大作业，绕不开的一个话题就是抄袭和魔改。

非常鄙视这样的同学。

我以前只觉得，从网上下载一个别人写好的项目改成自己的名字交上去已经是很不妥当了，今年更是了解到了刷新我的认知的事实——一个数据库大作业，在淘宝上不到 50 块钱就可以买来，加上报告也不到 100 块钱。

这么想来，从网上下载一个现成的项目，读一读源码，稍微看看懂，改一改，已经是很好的了。

6.7 还有很多可以改进

我的项目还有很多可以继续改进的地方。

一方面，我预留了各种接口，可以和财务系统、邮件发送系统挂钩，可以新增其他的表（例如更完整的权限表），以提供更丰富、灵活的功能。

另一方面。可以进一步加强用户体验，例如模糊搜索、全文匹配，以及，优化搜索效率。

附录

A git 提交记录

按 `commit` 时间倒序排列，最新的提交在最上面

```
/
├── 删除工号显示，因为 Vuex 不再存这个了
├── master
├── 修复公共查询点开 Tracking 仍会 401 的 BUG
├── MOCK 数据完成，准备写报告了
├── 如果当前城市和收件人城市相同，自动进入派件状态，并随机从派件员列表中指定一人派送（随机数取 Math.floor(Math.random() * count)）
├── 修复小错误
├── init 时创建超级管理员
├── 提供 init-database 脚本以供初始化数据库
├── 订单和接待员关系
├── 员工离职时（删除时），所属权限表 CASCADE，但是之前有过的订单记录从 RESTRICT 改为 SET_NULL
├── 特殊权限全部加了判断，以及该有异步的都异步了
├── 公共页面不经过前端路由、公共接口不做鉴权
├── 弃用 Permission 表
├── 特殊权限写入权限表
├── 物流信息
├── Welcome 页面
├── 成功后只清空包裹编号
├── 包裹编号输入回车直接提交，且成功后只清空包裹编号、不清空上面两个输入，这是为了方便今后引入扫描枪的时候可以批量自动提交
├── 录入一条新的物流信息后清空表单，以便连续填写（考虑到用户体验，没有直接刷新整个页面）
```

- 物流信息更新页面
- 只能查看分配给自己的、且正在派件中的订单。
- 未登录状态不会显示企业内部的子菜单
- 之前写的功能都放上主界面了
- 派件界面显示由谁派送、他的平均得分
- 修复新建时 `isSequence` 问题
- 哦, 不对, 修改的时候也要校验的, 是校验这个已经存在的邮箱是不是就是自己的, 不能是别人的
- 修复修改员工信息时会报邮箱已经存在的 bug, 该 `validate` 只应该在新增员工时检查
- 修改员工信息时异步获取邮箱
- 修改员工信息时异步获取员工权限
- 录入员工时选择权限并刷入不同的权限表
- 日期 `format`
- 物流表增加操作员
- 权限表
- 不同的部门表
- 打分后自动更新均分 (用 `orm` 实现的, 没有原生 `sql` 触发器语句)
- 打分
- 登录和用户中心分标签
- 优化删除时的用户体验, 修正可能会全部删完的 bug
- 表单 `validate` 逻辑修复和加强
- 所有的表单进行 `$ref[name].validate`
- 修复插入时因为 `async await` 的 403 406 问题
- 修改密码
- 派件功能
- 一些文字上的优化
- 完成了新建员工的功能 (没有引入权限时), 增加了邮箱唯一性的校验 (`oninputValidator`)
- 仓库 (在目前已有基础上完工, 后面会加入权限, 以及员工的部门, 那时还要修改这部分功能)
- 把城市做成了外键
- `WareHouse` 初步
- `validator` 初步
- 新增包裹的页面不会出现物流信息
- 修改包裹信息界面可以看到物流信息
- 修复修改成功后不跳转的问题, 修改 `PackageManage` 界面城市显示成数组的问题, 完整的分页功能
- `mounted` 请求完整城市数据, 修复修改包裹信息只显示第一级城市的问题
- 新建包裹和修改包裹信息
- 级联的选择器, 可筛选、可停在任意级别, 完整的异步查询已完成, 并可以提前以箭头标识是否还有子节点
- 城市列表
- 完整的拦截器

- lint
- 加强密码计算
- 引入 token 验证了
- 物流信息查看，并整合入包裹查询页面，用 Expand Row 做的
- 用 offset + limit 取数据
- 物流追踪初步
- 城市初步
- 包裹修改和新增的跳转逻辑及默认填充数据
- 真的包裹
- 添加部门的界面（假的
- 修复实例化组件的 warning
- 真的登录（md5(MD5_SUFFIX + password) 加密、后台数据库查询、结果响应）
- 学习 ORM 建表，并学习设置外键
- 配置 Sequelize
- Login 改成了 JWT 的
- 用 JWT 写了 Token 的假登录
- 假的数据库交互
- 维护依赖
- 路由时判断权限不足时自动跳转（又是一个大坑，有时间的话考虑用 JWT 重写这部分）
- 用 MenuItem.to 代替 router-link.to
- 回车键提交登录表单
- 假装有了登录与注销状态的切换
- vuex-electron 的那个大大大大坑
- 假装登录逻辑
- 包裹信息页面初步，同时封装了一些组件
- 修复所有 warning 和 error
- 分页初步
- 包裹管理页面初步
- 查询页面初步
- 用 Auth 完成权限认证页
- 拦截器初步
- 登录的前后端通信初步（缺拦截器）
- 假装登录界面
- 登录依赖
- 配置环境
- babel-plugin-transform-vue-jsx
- 记账页面初步
- 支出页面初步
- 钱包初步
- router 初步
- Card 初步
- iview-filter-table 复现
- 主要布局初步

```
|  
├─ init  
├─ 配置 iview 依赖  
├─ 使用 vue init simulatedgreg/electron-vue 重新初始化  
├─ electron-forge init -template=vue, 同时添加 pug, iview 等依赖  
└─ Initial commit
```

B 图片索引

图片索引

1	ER 图总览	11
2	员工角色相关 E-R 图	12
3	仓储 E-R 图	12
4	包裹及物流 E-R 图	13
5	派件 E-R 图	14
6	包裹与前台接待 E-R 图	14
7	公共查询	32
8	用户评分	33
9	用户登录	34
10	用户登录失败	34
11	用户登录成功	34
12	派件信息查看	35
13	运输状态录入	36
14	员工信息	37
15	员工信息维护	37
16	重置密码	38

17	搜索	38
18	包裹页面	39
19	物流跟踪	40
20	主要界面	41
21	Git 提交记录	52

C 表格索引

表格索引

1	Dispatchers	16
2	DispatchPairs	17
3	Employees	17
4	HumanResources	17
5	Locations	17
6	Members	18
7	PackageReceptionists	18
8	Packages	18
9	Receptionists	19
10	Trackings	19
11	Transports	19
12	WareHouseManagers	19
13	WareHouses	20
14	额外信息记录	20