**《宠物托运智能服务系统》**

**详细设计说明书：订单管理模块**

**课 程： 软件工程实训**

**队 伍： 深藏Blue**

**项目名称： GoGoPet**

**指导老师： 石秀金**

**成 员： 皮 超 221310417**

**赵鑫奕 221310109**

**张钰龙 221310131**

**李佳慧 221310203**

**丛梓璇 221310201**

**版本号V1.0**

**2025年4月5日**

目录

[1 引言 4](#_Toc194702885)

[1.1 编写目的 4](#_Toc194702886)

[1.2 项目概况 4](#_Toc194702887)

[1.3 术语定义 4](#_Toc194702888)

[1.4 参考资料 4](#_Toc194702889)

[2 系统概述 5](#_Toc194702890)

[2.1 系统架构 5](#_Toc194702891)

[2.2 模块功能 6](#_Toc194702892)

[3 程序设计详细描述 6](#_Toc194702893)

[3.1 订单管理模块 6](#_Toc194702894)

[3.1.1 创建订单功能 6](#_Toc194702895)

[3.1.2 在线支付功能 12](#_Toc194702896)

[3.1.3 查看订单功能 16](#_Toc194702897)

[3.1.1 更新订单功能 20](#_Toc194702898)

[3.2 表结构说明 22](#_Toc194702899)

[4 公用接口 24](#_Toc194702900)

[4.1 创建订单功能 24](#_Toc194702901)

[4.1.1 全局变量 24](#_Toc194702902)

[4.1.2 公用界面 25](#_Toc194702903)

[4.1.3 公用函数和过程 25](#_Toc194702904)

[4.1.4 公用表辞典 26](#_Toc194702905)

[4.2 在线支付功能 26](#_Toc194702906)

[4.2.1 全局变量 26](#_Toc194702907)

[4.2.2 公用界面 26](#_Toc194702908)

[4.2.3 公用函数和过程 27](#_Toc194702909)

[4.2.4 公用表辞典 27](#_Toc194702910)

[4.3 查看订单功能 28](#_Toc194702911)

[4.3.1 全局变量 28](#_Toc194702912)

[4.3.2 公用界面 28](#_Toc194702913)

[4.3.3 公用函数和过程 28](#_Toc194702914)

[4.3.4 公用表辞典 29](#_Toc194702915)

[4.4 更新订单功能 29](#_Toc194702916)

[4.4.1 全局变量 29](#_Toc194702917)

[4.4.2 公用界面 30](#_Toc194702918)

[4.4.3 公用函数和过程 30](#_Toc194702919)

[4.4.4 公用表辞典 31](#_Toc194702920)

1 引言

1.1 编写目的

本详细设计说明书用于指导**订单管理模块**的开发与实现，明确功能需求、交互流程及数据关联关系，为开发、测试及运维提供技术依据。本文档适用于开发人员、测试人员及相关项目成员。

1.2 项目概况

订单管理模块是宠物托运系统的核心模块，支持用户创建订单、在线支付、实时查看订单状态，以及托运公司更新订单进度。模块通过规范化流程提升托运效率，增强用户与托运方的信息透明度。

1.3 术语定义

表1 术语表

|  |  |
| --- | --- |
| 术语/缩写 | 定义 |
| 订单状态 | 订单生命周期中的阶段（已创建、已支付、已接单、运输中、已送达） |
| 托运方式 | 用户选择的运输类型（航空、陆运、专车等） |
| 支付状态 | 订单支付的完成情况（未支付、支付成功、支付失败） |
| 运输状态 | 宠物运输的实时进展（取件中、运输中、到达目的地） |

1.4 参考资料

* 《需求规格说明书》
* 《数据库设计说明书》
* 《社会交流模块详细设计说明书》

2 系统概述

2.1 系统架构

订单管理模块采用分层架构设计，前端基于 Vue3 + Element Plus 实现交互界面，后端基于 Spring Boot + MyBatis Plus 提供 RESTful API，数据库使用 MySQL 8.0+存储订单数据。关键组件如下：

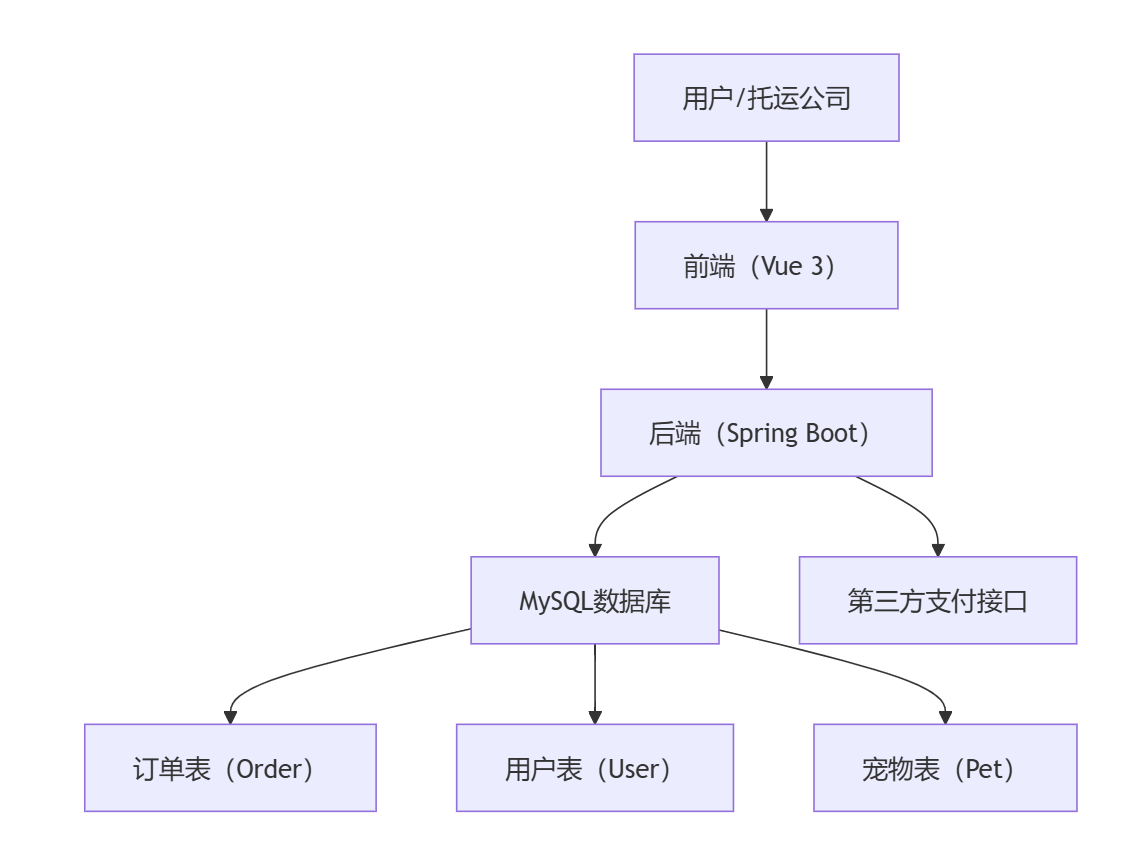
* 前端（Vue 3）：用户通过浏览器或移动端访问订单管理界面。
* 后端（Spring Boot）：处理订单创建、支付回调、状态查询等逻辑。
* 支付网关：集成支付宝/微信支付接口，支持在线支付。
* 消息队列（RabbitMQ）：异步处理订单状态更新，保障高并发场景下的系统稳定性。
* 数据库（MySQL）：存储订单表（t\_order）、用户表（t\_user）、宠物表（t\_pet）等数据。

图1 订单管理模块架构图

2.2 模块功能

* **创建订单**：用户填写宠物信息、托运方式、地址等生成订单。
* **在线支付**：支持支付宝/微信支付，支付成功更新订单状态。
* **查看订单**：用户查看订单进度及详情，托运公司查看完整订单信息。
* **更新订单**：托运公司修改订单状态及运输信息。

3 程序设计详细描述

3.1 订单管理模块

3.1.1 创建订单功能

表2 创建订单表

| 模块编号 | M\_DDGL\_1 | 源程序文件 | CreateOrder.vue |
| --- | --- | --- | --- |
| 功能 | 用户选择宠物、填写托运信息、上传图片，生成待支付订单。 | | |
| 输入参数 | - 用户ID（user\_id）：当前登录用户ID  - 宠物ID（pet\_id）：托运的宠物ID  - 托运地址（address）：目的地  - 托运时间（delivery\_time）：计划托运时间  - 图片ID（picture\_id）：上传的宠物照片ID | | |
| 要访问的表 | - t\_order（存储订单信息）  - t\_pet（校验宠物归属）  - t\_picture（存储上传的宠物图片） | | |
| 模块时序图 | PlantUML diagram  图2 创建订单功能时序图  时序图描述：  1.用户提交阶段   * 用户填写托运信息并提交 * 前端将数据发送至后端   2．后端验证阶段   * 验证用户是否有权限操作该宠物 * 校验必填字段完整性（如地址、时间）   3.订单生成阶段   * 生成唯一订单ID * 将订单数据写入数据库   4.最终反馈阶段   * 前端展示订单创建成功页面 * 用户可查看订单详情 | | |
| 输入界面 | 图3 输入页面效果图  主要控件说明：  1.宠物信息页   * 下拉选择框：用于选择宠物类型。 * 文本输入框：填写宠物名称、宠物年龄、宠物体重（kg），选填宠物描述。 * 按钮：“下一步”，提交当前页信息并跳转至联系方式页。   2.联系方式页   * 文本输入框：输入联系人姓名、联系电话、电子邮箱。 * 按钮：“上一步” 返回宠物信息页，“下一步” 跳转至托运信息页。   3.托运信息页   * 文本输入框：填写发货地址、收货地址，选填特殊要求。 * 单选按钮：选择托运方式。 * 日期选择框：选择托运日期。 * 复选框：勾选附加服务。 * 按钮：“上一步” 返回联系方式页，“下一步” 跳转至上传照片页。   4.上传照片页   * 文件上传组件：支持点击上传或拖放文件，限制格式为JPG/PNG，大小≤5MB。 * 文本显示：展示订单摘要。 * 按钮：“上一步” 返回托运信息页，“提交订单” 完成申请并提交。 | | |
| 输出界面 | 图4 输入页面效果图  主要控件说明：   * 状态图标：绿色对勾图标，直观提示“订单提交成功”状态。 * 标题文本：大字号标题“订单提交成功！”，辅助说明 “您的宠物托运订单已生成，等待支付”。 * 订单编号：静态文本显示订单号，附带创建时间。 * 标签：橙色“待支付”标签，标注订单当前状态。 * 支付信息框：含有提示文本和支付截止时间。 * 订单详情文本：托运方式、托运日期、发货地址、收货地址，均为静态文本展示。 * 宠物信息展示：宠物图片占位框，搭配文字描述。 * 费用明细：基础托运费和总计，以文本形式呈现费用信息。 | | |
| 出错处理 | 统一跳转至error.html页面 | | |

3.1.2 在线支付功能

表3 在线支付表

| 模块编号 | M\_DDGL\_2 | 源程序文件 | Payment.vue |
| --- | --- | --- | --- |
| 功能 | 用户完成支付后，系统接收回调并更新订单状态。 | | |
| 输入参数 | - 订单ID（order\_id）：需支付的订单ID - 支付方式（payment\_type）：支付宝/微信支付 | | |
| 要访问的表 | - t\_order（更新订单状态为“已支付”） - t\_payment（记录支付流水） | | |
| 模块时序图 | PlantUML diagram  图5 在线支付功能时序图  时序图描述：  1.用户支付：  User 向支付平台发起“扫描二维码完成支付”操作。  2.支付回调：  支付平台处理支付后，向系统发送“异步回调支付结果”。  3.系统处理：  系统接收回调后，先执行“验证签名”确保数据安全，验证通过后 “更新订单状态”。  4.通知用户：  系统完成订单状态更新后，调用短信/邮件服务，向 User 发送“短信/邮件通知支付成功”，最终用户接收通知，流程结束。 | | |
| 输入界面 | 图6 输入页面效果图  主要控件说明：   * 支付金额输入框：显示默认支付金额，用户可确认金额或修改，用于输入实际支付金额。 * 支付方式选项有支付宝和微信支付两种。 * 确认支付按钮：用户选择支付方式并确认金额后，点击此按钮提交支付请求，触发支付流程。 * 订单号显示：静态文本，展示订单号。 * 金额显示：静态文本，展示支付金额。 * 二维码图像:核心交互元素，用户通过支付宝扫描该二维码完成支付。 * 操作按钮:立即完成支付按钮，绿色主按钮，用户点击可触发支付流程；取消支付按钮，次要按钮，用户点击可取消当前支付操作。 | | |
| 输出界面 | 图7 输出页面效果图  主要控件说明：   * 状态文字：绿色的支付成功文字代表支付成功。 * 图标：转圈的图标表示要跳转到相应页面。 | | |
| 出错处理 | 统一跳转至error.html页面 | | |

3.1.3 查看订单功能

表4 查看订单表

| 模块编号 | M\_DDGL\_3 | 源程序文件 | OrderList.vue |
| --- | --- | --- | --- |
| 功能 | 用户和托运公司查看订单列表及详情。 | | |
| 输入参数 | - 用户ID（user\_id）：当前登录用户ID  - 宠物ID（pet\_id）：托运的宠物ID  - 托运地址（address）：目的地  - 托运时间（delivery\_time）：计划托运时间  - 图片ID（picture\_id）：上传的宠物照片ID  - 托运状态（states）：已接单、已取件、运输中、即将到达、已送达等状态  - 路线规划（route）：包括起点、终点、途经点 | | |
| 要访问的表 | - t\_order（存储订单信息）  - t\_pet（存储宠物信息）  - t\_driver（存储司机信息）  - t\_user（存储客户信息）  - t\_route（存储路线详情） | | |
| 模块时序图 | PlantUML diagram  图8 查看订单功能时序图  时序图描述：  1.用户发起请求：  用户点击“查看订单”并输入order\_id，触发查看操作。  2.前端请求发送：  前端（Vue 3）向后端（Spring Boot）发送请求，携带参数user\_id。  3.后端处理逻辑：   * 权限验证：后端验证user\_id与订单的关联关系，确保用户有权限查看。 * 数据库查询：查询t\_order表（通过order\_id），获取订单基础信息；查询t\_pet\_status表（通过pet\_id），获取宠物实时状态；查询t\_delivery\_log表（通过order\_id），获取运输日志。 * 数据整合：后端整合订单详情、宠物状态、运输日志，补充地图坐标、健康指标等信息。   4.响应与渲染：   * 后端将整合后的JSON数据返回给前端。 * 前端接收数据后，通过地图组件、状态面板、日志列表等渲染页面，最终向用户展示订单详情页。 | | |
| 输入界面 | 用户：    司机：    图9 输入页面效果图  主要控件说明：  用户：   * 订单编号：静态文本控件，展示订单编号。 * 下单时间：静态文本控件，显示下单时间。 * 托运方式：文本结合图标控件，显示托运方式。 * 预计到达时间：文本控件，展示预计到达时间。 * 状态节点：每个状态（已接单、已取件、运输中、即将到达、已送达）通过图标（圆形进度点）和文本呈现，现在处于的状态附带描述性文字。 * 司机头像图标：图形控件，展示司机头像。 * 司机姓名：文本控件，显示司机姓名及身份。   司机：   * 订单编号：展示订单编号，明确订单唯一标识。 * 地图占位图：显示“地图导航视图”“当前位置”等运输位置信息。 * 运输数据文本：剩余距离，告知剩余运输距离；预计驾驶时间，预估运输耗时；预计到达时间，显示具体时间。 * 进度条：显示已行驶距离和总距离。 * 宠物名称与类型：文本展示宠物名及类型。 * 健康状态标签：标签控件，显示健康状态。 * 特殊需求：文本控件，展示特殊需求内容。 * 路线规划模块：以列表形式展示运输路线，包含起点、途经点、终点。 * 联系人信息：文本显示联系人姓名、电话。 * 取件地址：文本展示取件地址。 * 联系客户按钮：交互按钮，点击可触发联系客户功能。 * 送达地址：文本显示送达地址。 * 预计到达时间：文本展示预计到达时间。 * 当前状态标签：标签控件，显示运输状态。 | | |
| 出错处理 | 统一跳转至error.html页面 | | |

3.1.1 更新订单功能

表5 更新订单表

| 模块编号 | M\_DDGL\_4 | 源程序文件 | UpdateOrder.vue |
| --- | --- | --- | --- |
| 功能 | 托运公司更新订单状态，并补充路线信息。 | | |
| 输入参数 | - order\_id（订单ID）  - status（订单状态）  - route（路线数据） | | |
| 要访问的表 | - t\_order（存储订单信息）  - t\_route（存储路线详情） | | |
| 模块时序图 | PlantUML diagram  图10 更新订单功能时序图  时序图描述：  1.员工操作与前端展示   * 托运公司员工登录前端管理后台，进入订单管理界面，前端显示待处理订单列表。 * 员工选择目标订单，输入新状态并补充路线坐标，触发前端向后端服务发送请求，携带参数{status, route, carrier\_id}。   2.后端处理逻辑   * 后端服务接收请求后，先执行两步校验：验证托运公司权限，校验路线坐标格式。 * 校验通过：更新数据库中t\_order表，设置订单新状态并添加路线坐标；向t\_delivery\_log表插入记录，记录状态变更；触发通知服务，通过用户 APP 推送状态变更通知；向后端返回 200 OK 响应，包含最新订单数据。 * 校验失败：返回错误响应，说明错误原因。   3.前端更新与用户通知   * 前端管理后台收到成功响应后，刷新界面显示更新后的订单状态，通过地图组件展示新路线。 * 通知服务异步触发短信/邮件，通知用户订单状态变更，流程结束。 | | |
| 输入界面 | 图11 输入页面效果图  主要控件说明：   * 下拉菜单（订单状态）：用于选择订单当前状态。 * 单选按钮（当前位置）：勾选选择宠物当前所在位置。 * 日期时间选择器（预计到达时间）：点击选择并设置订单预计到达时间。 * 文本输入框（备注信息）：输入关于订单状态更新的补充说明内容。 * 单选按钮（位置）：勾选设置路线点的具体位置。 * 日期时间选择器（到达时间）：选择并设置路线点的预计到达时间。 * 删除按钮：点击删除对应路线点信息。 * 添加路线点按钮：点击新增空白路线点，用于补充更多运输路线节点。 * 提交更新按钮：确认所有路线信息修改后，点击提交保存更新内容。 | | |
| 输出界面 | 图12 输出页面效果图  主要控件说明：   * 状态提示图标：绿色对勾图标，直观反馈操作结果（更新成功），强化视觉确认。 * 标题文本：“更新成功”标题，明确告知用户操作结果，起信息引导作用。 * 信息展示区：订单号、宠物名称、当前状态、更新时间等标签及对应内容，用于展示更新后的订单关键信息，帮助用户核对操作结果。 * 查看订单详情按钮：点击可跳转至该订单的详细信息页面，查看更全面的订单内容。 * 返回订单列表按钮：点击返回订单管理列表页面，便于继续管理其他订单。 * 返回首页按钮：点击返回系统首页，快速回到功能入口界面。 | | |
| 出错处理 | 统一跳转至error.html页面 | | |

3.2 表结构说明

本模块设计的数据库表主要为t\_order（存储订单信息）、t\_user（存储用户信息）、t\_pet（存储宠物信息）、t\_picture（存储上传的宠物图片），具体信息如下：

表6 用户表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 字段名 | 数据类型 | 允许空值 | 说明 |
| 用户ID | user\_id | VARCHAR(18) | N | 主键，唯一标识符 |
| 用户名 | user\_name | VARCHAR(63) | N | 注册信息 |
| 密码 | password | VARCHAR(63) | N | 注册信息 |
| 用户身份 | user\_type | CHAR(1) | N | 类型（用户/托运司机/管理员/……） |
| 用户地址 | user\_address | VARCHAR(63) | Y | 用户地址，限于用户、司机填写 |
| 用户头像 | picture\_id | VARCHAR(18) | Y | 外键，指明用户头像 |
| 公司ID | company\_id | VARCHAR(18) | Y | 外键，如果身份为司机则指明公司 |

表7 宠物表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 字段名 | 数据类型 | 允许空值 | 说明 |
| 宠物ID | pet\_id | VARCHAR(18) | N | 主键，唯一标识符 |
| 用户ID | user\_id | VARCHAR(18) | N | 外键，指明其主人 |
| 宠物名称 | pet\_name | VARCHAR(63) | N | 宠物信息 |
| 宠物品种 | pet\_breed | VARCHAR(63) | Y | 宠物信息 |
| 宠物体重 | pet\_weight | VARCHAR(18) | Y | 宠物信息 |
| 宠物健康状态 | pet\_health\_status | CHAR(1) | Y | 宠物信息 |

表8 图片表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 字段名 | 数据类型 | 允许空值 | 说明 |
| 图片ID | picture\_id | VARCHAR(18) | N | 主键，唯一标识符 |
| 图片用途 | picture\_usage | CHAR(1) | N | 标识其用途，包括（推文/图像识别/……），避免id重复时导致冲突 |

表9 订单表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 字段名 | 数据类型 | 允许空值 | 说明 |
| 订单ID | order\_id | VARCHAR(18) | N | 主键，唯一标识符 |
| 宠物ID | pet\_id | VARCHAR(18) | N | 外键，指明该订单涉及的宠物 |
| 图片ID | picture\_id | VARCHAR(18) | N | 外键，指明该订单包含的宠物图片 |
| 订单状态 | order\_status | CHAR(1) | N | 指明订单目前的状态（未指派/托运中/已完成/已取消/……） |
| 宠物当前状态 | pet\_status | CHAR(1) | N | 用于实时反馈当前托运宠物的状态，包括（健康/生病/……） |
| 运输状态 | delivery\_status | CHAR(1) | N | 指明当前托运司机的运输进度 |

4 公用接口

4.1 创建订单功能

4.1.1 全局变量

// 当前用户会话信息（通过ThreadLocal存储）

public class OrderContext {

    private static final ThreadLocal<Long> CURRENT\_USER\_ID = new ThreadLocal<>();

    public static void setCurrentUserId(Long userId) { CURRENT\_USER\_ID.set(userId); }

    public static Long getCurrentUserId() { return CURRENT\_USER\_ID.get(); }

}

// 运费计算配置

@Value("${delivery.base\_fee:50.00}")

private BigDecimal baseDeliveryFee; // 基础运费（单位：元）

@Value("${delivery.rate\_per\_km:1.20}")

private BigDecimal ratePerKilometer; // 每公里加价

4.1.2 公用界面

<!-- 地址选择组件（集成高德地图API） -->

<template>

  <div class="address-picker">

    <el-input

      v-model="address"

      placeholder="输入或选择地址"

      @click="showMapDialog"

    />

    <el-dialog v-model="mapVisible">

      <div id="map-container" style="height:400px"></div>

      <el-button @click="confirmAddress">确认选择</el-button>

    </el-dialog>

  </div>

</template>

<!-- 宠物选择表格组件 -->

<template>

  <el-table :data="pets" @row-click="handleSelectPet">

    <el-table-column prop="name" label="宠物名称"/>

    <el-table-column prop="healthStatus" label="健康状态"/>

  </el-table>

</template>

4.1.3 公用函数和过程

// 订单费用计算服务

@Service

public class OrderFeeCalculator {

    public BigDecimal calculateTotalFee(OrderRequest request) {

        // 1. 调用高德API计算距离

        BigDecimal distance = MapServiceClient.getDistance(

            request.getStartAddress(),

            request.getEndAddress()

        );

        // 2. 计算运费：基础费用 + 距离 \* 单价

        return baseDeliveryFee.add(distance.multiply(ratePerKilometer));

    }

}

// 订单表单校验工具

public class OrderValidator {

    public static void validateOrder(OrderRequest request) {

        if (request.getPets().stream().anyMatch(p -> p.getHealthStatus() != HealthStatus.HEALTHY)) {

            throw new BusinessException("PET\_HEALTH\_INVALID", "仅允许托运健康宠物");

        }

    }

}

4.1.4 公用表辞典

表10 创建订单公用表辞典

|  |  |  |  |
| --- | --- | --- | --- |
| **表名** | **字段名** | **格式** | **说明** |
| t\_order | order\_status | UNPAID（未支付）、PAID（已支付） | 订单生命周期状态 |
| t\_pet | health\_status | HEALTH、SICK、RECOVERING | 宠物健康状态校验依据 |
| t\_address | coordinate | POINT(lng lat)（MySQL地理数据类型） | 高德地图坐标标准化存储 |

4.2 在线支付功能

4.2.1 全局变量

// 支付网关配置（支付宝/微信）

@Value("${payment.alipay.app-id}")

private String alipayAppId;

@Value("${payment.wechat.mch-id}")

private String wechatMchId;

@Value("${payment.callback-url}")

private String paymentCallbackUrl;

4.2.2 公用界面

<!-- 支付方式选择组件 -->

<template>

  <div class="payment-methods">

    <el-radio-group v-model="selectedMethod">

      <el-radio label="ALIPAY"><img src="/icons/alipay.png"/></el-radio>

      <el-radio label="WECHAT"><img src="/icons/wechat.png"/></el-radio>

    </el-radio-group>

    <el-button @click="submitPayment">确认支付</el-button>

  </div>

</template>

4.2.3 公用函数和过程

// 支付服务（策略模式）

@Service

public class PaymentService {

    @Autowired

    private Map<String, PaymentStrategy> strategies; // ALIPAY/WECHAT实现类

    public PaymentResponse processPayment(PaymentRequest request) {

        return strategies.get(request.getMethod())

            .execute(request.getOrderId(), request.getAmount());

    }

}

// 支付回调验签工具

public class PaymentCallbackVerifier {

    public static boolean verifySignature(String payload, String sign, String method) {

        if ("ALIPAY".equals(method)) {

            return AlipaySignature.rsaCheck(payload, sign, alipayPublicKey);

        }

        // 微信验签逻辑...

    }

}

4.2.4 公用表辞典

表11 在线支付公用表辞典

|  |  |  |  |
| --- | --- | --- | --- |
| **表名** | **字段名** | **格式** | **说明** |
| t\_payment | payment\_method | ALIPAY、WECHAT、BANK\_TRANSFER | 支付渠道类型 |
| t\_payment | payment\_status | PENDING、SUCCESS、FAILED | 支付状态 |

4.3 查看订单功能

4.3.1 全局变量

// 地图渲染配置

@Value("${amap.key}")

private String amapJsApiKey;

@Value("${amap.route-color.user:#1890ff}")

private String userRouteColor; // 用户端路线颜色

@Value("${amap.route-color.driver:#52c41a}")

private String driverRouteColor; // 司机端路线颜色

4.3.2 公用界面

<!-- 订单详情卡片组件 -->

<template>

  <div class="order-card">

    <el-descriptions :column="2" border>

      <el-descriptions-item label="订单号">{{ order.id }}</el-descriptions-item>

      <el-descriptions-item label="托运状态">

        <el-tag :type="statusColor">{{ order.status }}</el-tag>

      </el-descriptions-item>

    </el-descriptions>

    <div id="order-map" style="height:300px;margin-top:20px"></div>

  </div>

</template>

4.3.3 公用函数和过程

// 数据权限过滤拦截器

public class OrderAccessInterceptor implements HandlerInterceptor {

    @Override

    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {

        Long orderId = Long.parseLong(request.getParameter("orderId"));

        // 用户只能查看自己订单，司机查看分配的订单

        if (!OrderService.hasAccess(orderId, UserUtils.getCurrentUser())) {

            response.sendError(403, "无权访问该订单");

            return false;

        }

        return true;

    }

}

// 地图路线渲染服务（前端）

export function renderOrderRoute(mapId, coordinates, role) {

    const map = new AMap.Map(mapId);

    const path = coordinates.map(c => new AMap.LngLat(c.lng, c.lat));

    const polyline = new AMap.Polyline({

        path: path,

        strokeColor: role === 'USER' ? '#1890ff' : '#52c41a',

        strokeWeight: 4

    });

    map.add(polyline);

    map.setFitView();

}

4.3.4 公用表辞典

表12 查看订单公用表辞典

|  |  |  |  |
| --- | --- | --- | --- |
| **表名** | **字段名** | **格式** | **说明** |
| t\_delivery\_log | event\_type | PICKUP（揽收）、IN\_TRANSIT（运输中） | 运输事件类型 |
| t\_order | visibility | USER\_VISIBLE、DRIVER\_VISIBLE | 数据可见性控制 |

4.4 更新订单功能

4.4.1 全局变量

// 路线更新规则配置

@Value("${delivery.route.max-points:50}")

private int maxRoutePoints; // 单次更新最多添加50个坐标点

@Value("${delivery.route.update-interval:300000}")

private long minUpdateInterval; // 至少5分钟更新一次（毫秒）

4.4.2 公用界面

<!-- 路线坐标编辑器组件 -->

<template>

  <div class="route-editor">

    <el-button @click="openMapEditor">在地图上绘制路线</el-button>

    <el-tag v-for="(point, index) in points" :key="index">

      {{ point.lng }}, {{ point.lat }}

    </el-tag>

  </div>

</template>

4.4.3 公用函数和过程

// 路线标准化服务

@Service

public class RouteService {

    public List<GeoPoint> normalizeRoute(List<GeoPoint> rawPoints) {

        // 1. 过滤无效坐标（超出服务区域）

        // 2. 使用Douglas-Peucker算法简化路径

        // 3. 确保坐标点按时间顺序排列

        return simplifiedPoints;

    }

}

// 订单状态机校验

public class OrderStateMachine {

    public static void validateTransition(OrderStatus oldStatus, OrderStatus newStatus) {

        Map<OrderStatus, List<OrderStatus>> allowedTransitions = Map.of(

            OrderStatus.PAID, List.of(OrderStatus.PICKED\_UP),

            OrderStatus.PICKED\_UP, List.of(OrderStatus.IN\_TRANSIT, OrderStatus.DELAYED),

            // ...其他状态流转规则

        );

        if (!allowedTransitions.get(oldStatus).contains(newStatus)) {

            throw new IllegalStateException("状态变更非法");

        }

    }

}

4.4.4 公用表辞典

表13 更新订单公用表辞典

|  |  |  |  |
| --- | --- | --- | --- |
| **表名** | **字段名** | **格式** | **说明** |
| t\_order\_route | route\_type | MAIN（主路线）、DETOUR（绕行路线） | 路线类型 |
| t\_order | update\_lock | 版本号（@Version） | 乐观锁防止并发更新冲突 |