**《宠物托运智能服务系统》**

**详细设计说明书：托运追踪模块**

**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7日**

## 引言

### 编写目的

本详细设计说明书用于指导托运追踪模块的开发与实现，明确模块的功能需求、技术实现、数据关联关系等，为开发、测试及运维团队提供技术依据。本文档适用于开发人员、测试人员以及相关项目成员，帮助他们理解模块的设计细节并确保高质量的实现。

### **项目概述**

托运追踪模块是宠物托运系统中的核心功能之一，旨在提高托运过程的透明度与安全性。模块结合照片上传、智能识别、订单进度反馈，宠物状态分析等技术，为宠物主人提供全面的宠物托运跟踪体验。用户可以通过订单状态和托运司机反馈得知宠物的状态，确保宠物的健康和托运效率。通过这一模块，宠物主人可以有效减少不安和焦虑，提升用户体验和服务质量。

### **术语定义**

|  |  |
| --- | --- |
| **术语/缩写** | **定义** |
| 托运追踪 | 实时查看宠物托运状态，包括照片、视频及定位等信息。 |
| 照片/视频上传 | 托运公司定期上传宠物的照片或视频，供用户查看。 |
| 宠物状态分析 | 通过图像识别技术分析宠物的健康状况。 |
| 地图定位 | 实时显示宠物的运输位置，并提供路线规划及异常报警。 |

### **参考资料**

· 《需求规格说明书》

· 《数据库设计说明书》

· 《宠物托运系统详细设计说明书》

## 模块概述

### 2.1、模块架构

托运追踪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 提供 RESTful API。模块的关键技术组件如下：

· 前端（Vue）：用户通过网页访问订单追踪界面，查看照片和托运状态。

· 后端（Spring Boot + MyBatis）：负责处理上传照片和订单进度、宠 物状态等服务。

· 图片处理系统：对上传的照片进行处理和存储，用于反馈宠物状态

· 数据库（MySQL）：存储与托运相关的所有数据，包括订单信息、位置 数据、图片等。

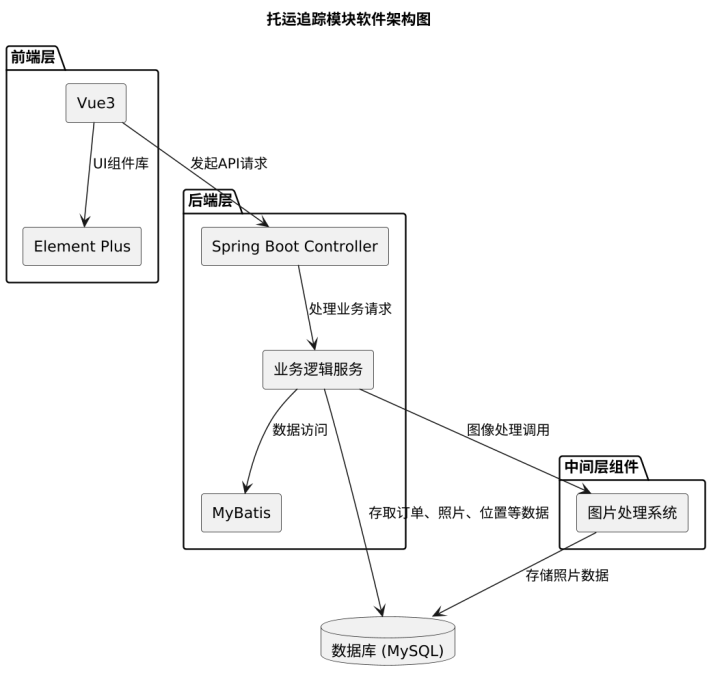


图1 托运追踪模块架构图

### 2.2、模块功能

· 更新订单：托运司机在托运图中的固定点位刷新进度，并上传宠物状态

· 宠物状态分析：使用动物情绪识别，动物状态识别的模型分析宠物状态

· 宠物跟踪模块：利用固定点位的刷新，使用较低频度显示宠物当前位置

## **程序设计详细描述**

| **模块编号** | M\_DDGLMK\_1 | **源程序文件** | OrderTrackingController.vue OrderTrackingController.java  OrderService.java |
| --- | --- | --- | --- |
| **功能** | 托运司机在系统规定的运输节点更新订单进度，并上传对应宠物的实时状态信息（如照片、视频）。系统接收后存入数据库，并通知用户。 | | |
| **输入参数** | 订单表（T\_Order）与宠物状态信息、上传的图片或视频 | | |
| **要访问的表** | **订单表（T\_Order）、宠物表（T\_Pet）、图片信息表（T\_Media）** | | |
| **模块时序图** | IMG_256  图：时序图 | | |
| **输入界面** | 首页为浏览输入页，如下图：  IMG_256 | | |
| **模块编号** | M\_CWFXMK\_1 | **源程序文件** | PetStatusAnalyzer.java ImageProcessingService.java |
| **功能** | 系统对上传的宠物照片或视频进行情绪及健康状态识别，分析后形成报告反馈给宠物主人。 | | |
| **输入参数** | 图片/视频媒体内容，宠物特征模型数据 | | |
| **要访问的表** | 宠物表（T\_Pet） | | |
| **模块时序图** | IMG_256  图：时序图 | | |
| **输入界面** | 首页为浏览输入页，如下图： | | |

| **模块编号** | M\_GZMK\_1 | **源程序文件** | TrackingController.java PointService.java |
| --- | --- | --- | --- |
| **功能** | 系统周期性接收托运司机上报的点位位置信息，在用户端以地图形式呈现宠物的当前位置与路线轨迹。 | | |
| **输入参数** | 图片/视频媒体内容，宠物特征模型数据 | | |
| **要访问的表** | 宠物表（T\_Pet） | | |
| **模块时序图** | IMG_256  图：时序图 | | |
| **输入界面** | 首页为浏览输入页，如下图： | | |

## **公用接口**

### **4.1、更新订单** 4.1.1 全局变量

// 当前托运进度和位置（通过ThreadLocal存储）

public class OrderContext {

private static final ThreadLocal<String> CURRENT\_ORDER\_STATUS = new ThreadLocal<>();

private static final ThreadLocal<List<GeoPoint>> CURRENT\_ROUTE\_POINTS = new ThreadLocal<>();

public static void setCurrentOrderStatus(String status) { CURRENT\_ORDER\_STATUS.set(status); }

public static String getCurrentOrderStatus() { return CURRENT\_ORDER\_STATUS.get(); }

public static void setCurrentRoutePoints(List<GeoPoint> points) { CURRENT\_ROUTE\_POINTS.set(points); }

public static List<GeoPoint> getCurrentRoutePoints() { return CURRENT\_ROUTE\_POINTS.get(); }

}

4.1.2 公用界面

<!-- 更新订单界面（展示运输进度） -->

<template>

<div class="order-progress">

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

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

<el-descriptions-item label="当前状态">

<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>

<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

// 更新订单进度和状态

@Service

public class OrderProgressService {

@Autowired

private RouteService routeService;

public void updateOrderProgress(Long orderId, List<GeoPoint> updatedRoutePoints) {

// 1. 更新订单位置

List<GeoPoint> normalizedPoints = routeService.normalizeRoute(updatedRoutePoints);

OrderContext.setCurrentRoutePoints(normalizedPoints);

// 2. 更新订单状态为运输中

OrderContext.setCurrentOrderStatus("IN\_TRANSIT");

}

}

4.1.4 公用表辞典

| **表名** | **字段名** | **格式** | **说明** |
| --- | --- | --- | --- |
| t\_order | order\_status | UNPAID、PAID、IN\_TRANSIT、DELIVERED | 订单生命周期状态 |
| t\_order\_route | route\_type | MAIN、DETOUR | 路线类型 |

**4.2 宠物状态分析功能**

4.2.1 全局变量

@Value("${emotion.model.path}")

private String emotionModelPath; // 模型路径配置

4.2.2 公用界面

<!-- 宠物状态展示组件 -->

<template>

<div class="pet-status">

<el-tag :type="healthStatusColor">{{ petStatus }}</el-tag>

</div>

</template>

4.2.3 公用界面

// 宠物状态分析服务

@Service

public class PetStatusAnalysisService {

public PetStatus analyzePetEmotion(Long petId) {

// 1. 使用模型分析宠物情绪

Pet pet = petRepository.findById(petId);

String petEmotion = EmotionModel.predict(pet.getImagePath());

// 2. 更新宠物健康状态

if ("SICK".equals(petEmotion)) {

pet.setHealthStatus(HealthStatus.SICK);

} else {

pet.setHealthStatus(HealthStatus.HEALTHY);

}

petRepository.save(pet);

return pet.getHealthStatus();

}

}

4.2.4 公用表辞典

| **表名** | **字段名** | **格式** | **说明** |
| --- | --- | --- | --- |
| t\_pet | health\_status | HEALTH、SICK、RECOVERING | 宠物健康状态 |

**4.3 宠物跟踪功能\***

4.3.1 全局变量

@Value("${tracking.update-interval:3600000}")

private long updateInterval; // 每小时更新一次位置

4.3.2 公用界面

<!-- 宠物位置展示组件 -->

<template>

<div class="pet-tracking">

<el-button @click="trackPet">实时跟踪</el-button>

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

<el-descriptions-item label="当前坐标">{{ currentPosition }}</el-descriptions-item>

</el-descriptions>

</div>

</template>

4.3.3 公同用函数和过程

// 宠物跟踪服务

@Service

public class PetTrackingService {

public void trackPet(Long petId) {

// 获取宠物当前位置

Pet pet = petRepository.findById(petId);

String currentLocation = pet.getCurrentLocation(); // 获取宠物最新位置

// 更新前端展示的坐标

PetContext.setCurrentPosition(currentLocation);

}

}

4.3.4 公用辞典表

| 表**名** | **字段名** | **格式** | **说明** |
| --- | --- | --- | --- |
| t\_pet | current\_location | POINT(lng lat) | 宠物当前位置 |