**《宠物托运智能服务系统》**

**概要设计说明书**

**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3月31日**

**目录**

[1 引言 3](#_Toc194339218)

[1.1编写目的 3](#_Toc194339219)

[1.2项目概况 3](#_Toc194339220)

[1.3术语定义 3](#_Toc194339221)

[1.4 参考资料 3](#_Toc194339222)

[2 系统概述 4](#_Toc194339223)

[2.1 运行环境 4](#_Toc194339224)

[2.1.1 硬件环境 4](#_Toc194339225)

[2.1.2 软件环境 4](#_Toc194339226)

[2.1.3 数据结构 5](#_Toc194339227)

[2.2 需求概述 6](#_Toc194339228)

[3 原则与规范 6](#_Toc194339229)

[3.1 设计原则 6](#_Toc194339230)

[3.2 设计规范 7](#_Toc194339231)

[3.2.1 命名规范 7](#_Toc194339232)

[3.2.2 界面规范 7](#_Toc194339233)

[3.2.3 代码规范 8](#_Toc194339234)

[3.2.4 数据库规范 8](#_Toc194339235)

[3.2.5 日志规范 8](#_Toc194339236)

[3.2.6 AI服务规范 8](#_Toc194339237)

[4. 总体设计 9](#_Toc194339238)

[4.1 系统划分及功能描述 9](#_Toc194339239)

[4.2系统架构模型和实现样例 9](#_Toc194339240)

[4.2.1 系统架构模型 9](#_Toc194339241)

[4.2.2 实现样例 10](#_Toc194339242)

[4.3 公用接口程序设计说明 10](#_Toc194339243)

[4.3.1 全局变量 10](#_Toc194339244)

[4.3.2 其他 11](#_Toc194339245)

1 引言

* 1. 编写目的

本概要设计说明书旨在明确**宠物托运智能服务系统**的总体架构设计、功能模块划分及技术实现方案，为后续详细设计、开发、测试和部署提供指导依据。

1.2项目概况

本系统面向**宠物主人、托运公司及平台管理方**，提供智能化的宠物托运服务管理功能，核心功能包括：用户管理、订单管理、托运追踪、社会交流及智能客服模块，通过数字化手段提升托运效率与安全性。

1.3术语定义

表1 术语表

|  |  |
| --- | --- |
| 术语/缩写 | 定义 |
| GPS | 全球定位系统（Global Positioning System），用于实时跟踪宠物托运状态。 |
| RAG | 检索增强生成（Retrieval-Augmented Generation），用于智能客服解答托运问题。 |
| API | 应用程序接口（Application Programming Interface），用于系统功能的外部调用。 |

1.4 参考资料

《系统实施计划》

《需求规格说明书-V1.0》

2 系统概述

2.1 运行环境

2.1.1 硬件环境

（1）服务器配置

* CPU：8核及以上
* 内存：16GB及以上
* 硬盘：500GB SSD及以上
* 带宽：10Mbps及以上

（2）客户端设备

* 移动端：Android 8.0+/iOS 12+
* PC端：Windows 10+/macOS 10.15+
* 平板端：iPadOS 14+/Android 10+

2.1.2 软件环境

（1）后端

* 开发语言：Java 17
* 核心框架：Spring Boot 3.0 + MyBatis Plus
* 接口协议：RESTful API
* 安全认证：JWT实现用户身份鉴权
* 缓存服务：Redis 6.0+ 进行高频数据缓存
* 容器化部署：Docker + Nginx进行反向代理与负载均衡

（2）前端

* 核心框架：Vue 3 + Vite
* UI 组件库：Ant Design Vue统一组件风格
* 网络请求：Axios封装RESTful接口调用
* 状态管理：Pinia替代Vuex，实现轻量级状态管理

（3）数据库

* 主数据库：MySQL 8.0
* 数据库工具：MyBatis Plus支持动态 SQL
* 文件存储：阿里云 OSS

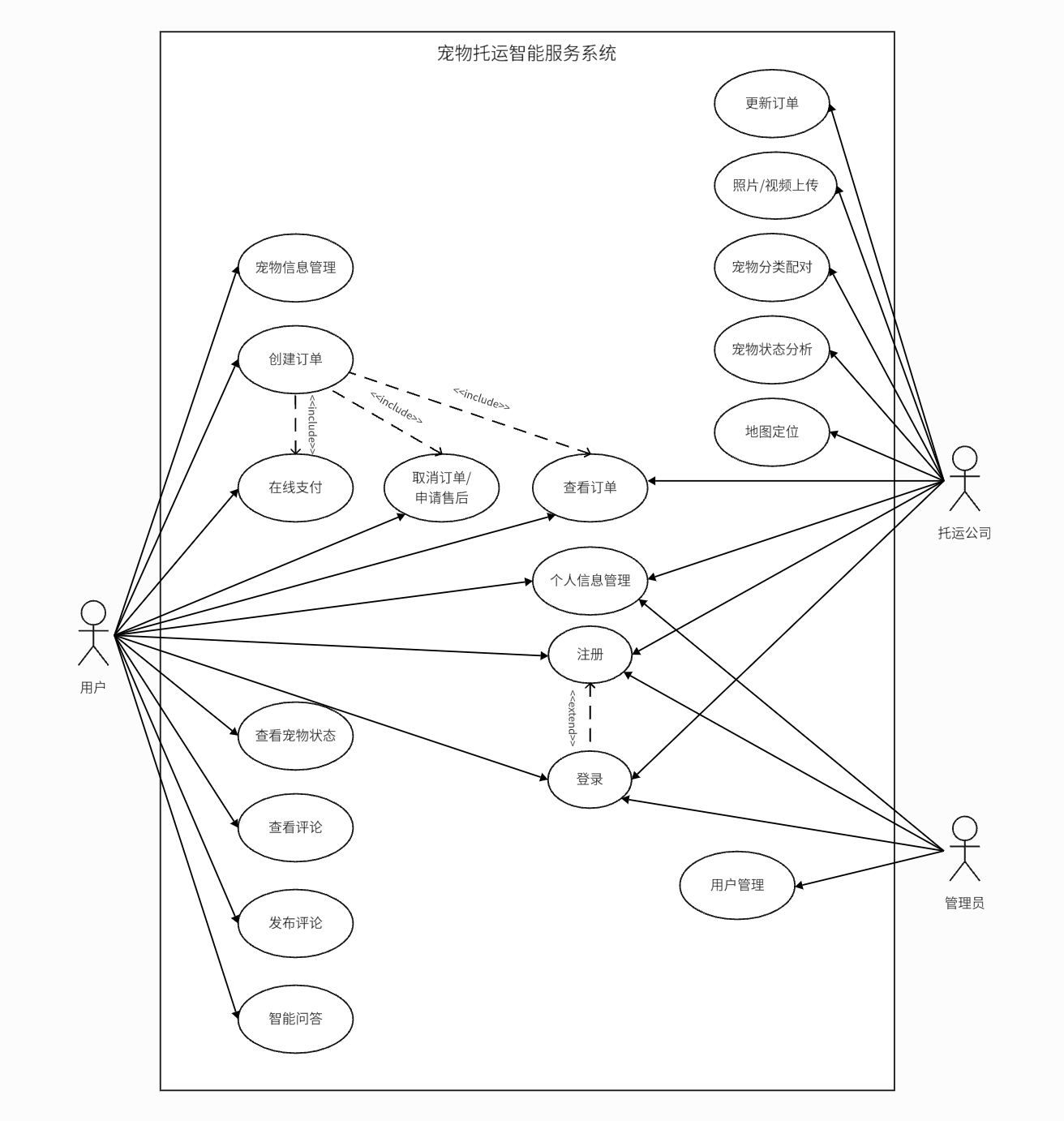
（4）部署方案

* 容器化：Docker打包后端服务与前端静态资源
* 代理服务器：Nginx配置反向代理、静态资源托管与HTTPS加密

2.1.3 数据结构

核心数据表

* t\_user：用户表（用户ID，用户名，密码，用户身份，用户地址，用户头像，公司ID）
* t\_pet：宠物表（宠物ID，用户ID，宠物名称，宠物品种，宠物体重，宠物健康状态）
* t\_picture：图片表（图片ID，图片用途）
* t\_order：订单表（订单ID，宠物ID，图片ID，订单状态，托运状态，运输状态）
* t\_chat：对话表（对话ID，用户ID，对话内容）
* t\_post：推文表（推文ID，用户ID，推文内容，推文评论）
* t\_company：公司表（公司ID，公司介绍，公司地址）
* t\_comment：评论表（评价ID，公司ID，用户ID，评论内容）

2.2 需求概述

图：系统用例图

3 原则与规范

3.1 设计原则

（1）可靠性

* 采用主备数据库+负载均衡架构，确保核心业务（下单、支付、追踪）可用性≥99.9%。
* 关键服务（如智能AI、地图定位）实现故障自动切换，保障系统稳定性。

（2）安全性

* 用户数据传输加密（SSL/TLS），存储加密（AES-256），敏感信息双重加密。
* 基于角色的访问控制（RBAC），限制未授权操作，防范SQL注入、XSS等攻击。

（3）可扩展性

* 模块化架构设计，支持动态扩展功能（如新增AI模型、第三方服务对接）。
* 数据库采用分库分表策略，支持未来100万+订单数据存储。

（4）用户体验优化

* 界面响应时间≤2 秒（如订单创建、AI问答），地图定位实时刷新。
* 移动端优先适配，支持Android/iOS设备的交互优化。

（5）合规性

* 遵循《动物防疫法》《道路运输管理条例》，确保托运流程合法合规。
* 用户数据处理符合GDPR和国内隐私保护法规。

3.2 设计规范

3.2.1 命名规范

* 模块命名：采用英文单词组合，如UserManagement、OrderTracking。
* 数据库表名：下划线分隔小写字母，如user\_info、pet\_order。
* 字段命名：驼峰式命名，如userId、createTime。
* 接口命名：RESTful 风格，如GET/api/orders/{id}。

3.2.2 界面规范

* 布局统一：PC端采用左侧导航栏+顶部工具栏，移动端采用底部标签栏。
* 交互反馈：操作成功/失败提示统一使用Toast或模态框，避免阻塞式弹窗。
* 响应式设计：支持平板、手机等多设备自适应，分辨率适配1080P及以上。

3.2.3 代码规范

（1）后端（Java）

* 遵循Spring Boot编码规范，类名首字母大写，方法名小写驼峰。
* 避免复杂逻辑嵌套，单个方法代码量≤200行。
* 使用Lombok简化代码（如@Data、@Slf4j）。

（2）前端（Vue.js）

* 组件命名PascalCase，如PetInfo.vue。
* 样式采用SCSS预处理器，统一主题色。
* 路由配置按需加载，减少首屏加载时间。

3.2.4 数据库规范

* 索引优化：为高频查询字段（如orderStatus、userId）添加索引。
* 事务控制：订单创建、支付等操作使用分布式事务。
* 分库策略：按业务垂直分库（用户库、订单库、日志库）。

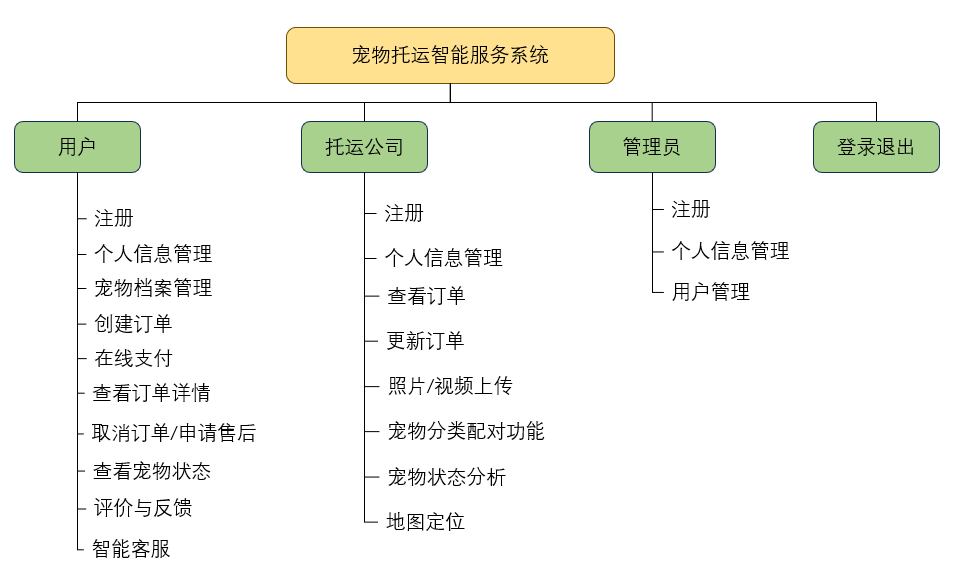
3.2.5 日志规范

* 使用ELK Stack集中管理日志，记录关键操作（如支付、订单修改）。
* 日志级别分级：ERROR记录异常，INFO记录业务流程，DEBUG用于开发调试。
* 日志保留期≥6个月，重要日志长期归档。

3.2.6 AI服务规范

* LLM模型推理响应时间≤2秒，支持异步处理高并发请求。
* RAG知识库每日更新，确保宠物托运政策、疾病处理等信息时效性。
* 图像识别模型准确率≥95%，定期通过测试集验证。

4. 总体设计

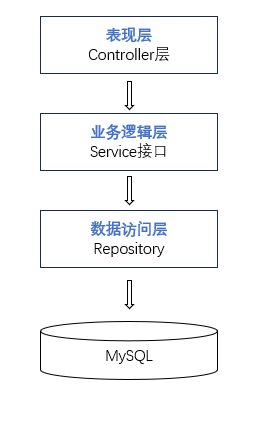
4.1 系统划分及功能描述

4.2系统架构模型和实现样例

4.2.1 系统架构模型

采用**分层架构**，基于SpringBoot+Spring MVC实现前后端分离设计：

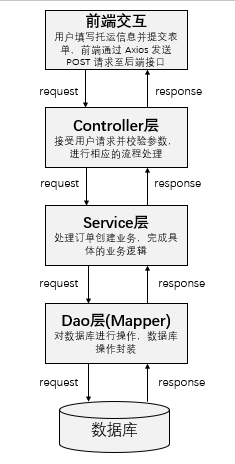
前端采用Vue.js3+Vite，后端Spring Boot3.x + Spring MVC，数据库采用MySQL



* Controller 接收请求，返回 JSON 响应
* 使用 @RestController 注解定义 RESTful API 接口
* Service 实现核心业务逻辑
* 调用基础设施层集成第三方服务（如高德地图 API、支付宝接口）
* 使用MyBatis操作数据库

4.2.2 实现样例

以**订单创建流程**为例，详细说明各层交互逻辑：



4.3 公用接口程序设计说明

4.3.1 全局变量

全局变量在系统中应严格限制使用，仅在必要时通过单例模式或依赖注入管理共享资源，避免数据竞争与副作用。

1. 全局配置参数

**定义**：系统级配置（如第三方API密钥、超时时间）。

**实现：**

@Configuration  
@ConfigurationProperties(prefix = "system.config")  
public class GlobalConfig {  
 private String amapApiKey; // 高德地图API密钥   
 private int requestTimeout; // HTTP请求超时时间（毫秒）   
 // Getter/Setter   
}

**访问方式：**通过Spring @Autowired 注入到需使用的类中。

1. 共享服务实例

**定义：**需要全局访问的服务。

**实现方式：**

@Component  
public class RedisManager {  
 @Autowired  
 private RedisTemplate<String, String> redisTemplate;  
  
 // 全局缓存操作接口   
 public void setCache(String key, String value) {  
 redisTemplate.opsForValue().set(key, value);  
 }  
}

4.3.2 其他

1.公共工具类：

**功能：**封装通用操作逻辑，避免代码冗余。

**示例：**

public class DateUtils {  
 private static final SimpleDateFormat *sdf* = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
  
 // 日期格式化   
 public static String formatDate(Date date) {  
 return *sdf*.format(date);  
 }  
}  
  
public class EncryptionUtils {  
 // AES加密   
 public static String encrypt(String data, String key) { ... }  
}

2.全局异常处理：

**功能：**统一捕获并处理系统异常，返回标准化错误响应。

**实现方式：**

@ControllerAdvice  
public class GlobalExceptionHandler {  
 @ExceptionHandler(Exception.class)  
 public ResponseEntity<ErrorResponse> handleException(Exception ex) {  
 ErrorResponse response = new ErrorResponse(500, "系统内部错误", ex.getMessage());  
 return ResponseEntity.status(500).body(response);  
 }  
  
 // 自定义业务异常处理   
 @ExceptionHandler(BusinessException.class)  
 public ResponseEntity<ErrorResponse> handleBusinessException(BusinessException ex) {  
 ErrorResponse response = new ErrorResponse(ex.getCode(), ex.getMessage(), null);  
 return ResponseEntity.status(400).body(response);  
 }  
}

1. 日志记录接口

**功能：**统一日志格式、级别及存储策略。

**实现方式：**

@Slf4j  
@Component  
public class SystemLogger {  
 // 记录操作日志（存入数据库或文件）   
 public void logOperation(String userId, String action) {  
 log.info("用户 {} 执行操作：{}", userId, action);  
 // 可扩展为异步写入数据库   
 }  
  
 // 记录错误日志（带堆栈信息）   
 public void logError(String module, Exception e) {  
 log.error("模块 {} 发生异常：{}", module, e.getMessage(), e);  
 }  
}

1. 权限验证模块

**功能：**统一管理用户鉴权与角色权限。

**实现方式：**

@Component  
public class AuthValidator {  
 @Autowired  
 private JwtTokenProvider jwtTokenProvider;  
  
 // 验证Token并获取用户角色   
 public UserRole validateToken(String token) {  
 if (!jwtTokenProvider.validateToken(token)) {  
 throw new BusinessException("无效的Token");  
 }  
 return jwtTokenProvider.getRoleFromToken(token);  
 }  
  
 // 校验用户操作权限   
 public void checkPermission(UserRole role, String requiredPermission) {  
 if (!role.getPermissions().contains(requiredPermission)) {  
 throw new BusinessException("权限不足");  
 }  
 }  
}

1. 第三方服务封装接口

**功能：**统一封装高德地图、支付宝等第三方服务调用逻辑。

**示例：**

@Component  
public class MapServiceClient {  
 @Autowired  
 private RestTemplate restTemplate;  
 @Value("${amap.api.url}")  
 private String apiUrl;  
  
 // 调用高德地图路径规划   
 public Route calculateRoute(String start, String end) {  
 String url = apiUrl + "?origin=" + start + "&destination=" + end;  
 return restTemplate.getForObject(url, Route.class);  
 }  
}