软件开发技术方案：

**1.项目背景与需求分析**

* 1. **背景介绍**

随着AI大模型的多样化与应用场景的快速拓展，企业内部对于“多模型能力调用”、“角色任务适配”、“多模态交互”提出了更高要求。本产品旨在为企业打造一个**统一的多模型智能对话平台**，提升办公效率与内容生产能力。

### **1.2 产品定位**

一个集成多AI模型（ChatGPT、DeepSeek、通义千问等）的对话式交互平台，支持多模态输入，具备角色适配、提示词管理和对话内容归档能力。

**1.3 产品预期**

本项目预期可实现以下功能:

1.打通多模型接入能力，后台可自由配置API。

2.前端支持图文声像输入，打造全渠道对话入口。

3.提供账号、角色、权限、提示词绑定的完整闭环。

4.管理端可审查所有对话内容，确保内容安全与合规。

**2．技术架构与设计**

### **2.1茵浪技术架构图**

|  |  |  |  |
| --- | --- | --- | --- |
| **模块** | **原文档技术栈** | **调整后技术栈** | **修改说明** |
| **前端框架** | React，Tailwind CSS | **保留** Vue CLI + Element UI | 前端无需改动，保持现有开发进度。 |
| **后端框架** | Node.js/Express 或 Django | **替换为** Spring Boot 3.x | 使用Java生态，适合企业级开发和高性能场景。 |
| **ORM框架** | Django ORM / Sequelize | **替换为** MyBatis-Plus | MyBatis-Plus支持代码自动生成、Lambda查询、分页插件，显著提升开发效率。 |
| **主数据库** | MySQL MongoDB | **优化为** MySQL（主库） + MongoDB（可选） | 使用MySQL存储结构化数据（用户、权限、日志），MongoDB仅用于非结构化会话历史（若需保留多模态数据）。 |
| **模型接口封装** | Node.js HTTP模块 / Python Requests | **替换为** Spring Boot RestTemplate/WebClient | 直接调用多模型API，简化微服务依赖。 |
| **文件存储** | 阿里云OSS / MinIO | **保留** 阿里云OSS / 本地存储 | 根据需求选择，初期可用本地存储（Nginx代理）。 |
| **权限管理** | 自定义实现 / Django Admin | **替换为** Spring Security + Sa-Token | Sa-Token提供开箱即用的鉴权模块，支持角色绑定、权限拦截。 |
| **代码生成** | 手动编写 | **新增** MyBatis-Plus代码生成器 | Sa-Token提供开箱即用的鉴权模块，支持角色绑定、权限拦截。 |
| **部署方案** | Node.js PM2 / Django + Gunicorn | **替换为** Spring Boot内嵌Tomcat + Docker | 使用Docker Compose打包MySQL + Spring Boot ，简化部署流程。 |

**2.2 前端功能（PC+移动端自适应）**

| **模块** | **功能点** |
| --- | --- |
| 登录/注册 | 管理员创建账，角色绑定 |
| 对话主界面 | 支持文字、语音、图片、视频上传，多模型选择器 |
| 响应展示区 | 显示模型名称、响应内容、时间戳、复制、收藏等 |
| 会话历史 | 按时间/标签查看过往记录，支持删除、归档 |
| 模型选择器 | 自由选择配置好的AI模型，并动态切换 |
| 设置页面 | 设定默认模型、消息保存周期、隐私设置等 |

### **2.3 后端功能**

| **模块** | **功能点** |
| --- | --- |
| 用户系统 | 角色分配（管理员/普通用户） |
| 模型接入 | 支持后台添加模型API，配置模型名、描述、调用地址、key |
| 角色-Prompt绑定 | 可针对不同角色配置提示词模版，提高调用效率 |
| 对话存档 | 所有交互内容本地化保存，支持管理员查询审计 |
| 行为日志 | 所有操作行为记录，便于风控排查 |

### **2.4用户角色与权限**

| **角色** | **权限范围** |
| --- | --- |
| 超级管理员 | 所有系统权限，包括模型配置、日志查看、敏感词设置，管理用户、角色、提示词配置，查看内容记录等 |
| 普通用户 | 对话使用权限，查看/导出自己的对话记录 |

1. **技术实现与详细设计**

**3.1具体的技术实现**

**(数据库ER图，接口文档，数据库权限逻辑等等，开发过程中补充。。。。。。)**

### **3.2界面设计与交互流程**

登录页：由管理员创建并分配账号

首页：展示品牌Logo、欢迎语、模型快速入口

对话页：用户输入——>选择模型——>展示AI响应

设置页：偏好设置、模型默认值设定

管理后台：模型管理、角色设置、日志审查等

**4.测试方案**

### **4.1 测试目标**

确保系统功能完整性、多模型接口稳定性、权限控制严格性及数据安全性，满足以下核心需求：

1.多模态输入与多模型动态切换功能正常。

2.用户角色与权限绑定逻辑无误。

3.日志审计模块有效拦截风险。

### **4.2 测试范围**

| **测试类型** | **测试内容** |
| --- | --- |
| **功能测试** | **- 用户登录/注册、角色绑定 - 多模型选择与响应展示（文字/图片/音视频） - 会话历史归档与删除** |
| **接口测试** | **- 模型API调用成功率（含Header/Token校验） - 跨模型响应格式统一性（JSON/XML适配）** |
| **性能测试** | **- 单接口压测（JMeter模拟1000并发） - 混合负载下MySQL与MongoDB读写性能（TPS/QPS监控）** |
| **安全测试** | **- SQL注入/XSS攻击防御验证 - 敏感词过滤覆盖率（内置词库+自定义词库） - 管理员操作日志防篡改机制** |
| **兼容性测试** | **- 浏览器（Chrome/Firefox/Safari） - 移动端（iOS/Android系统版本适配）** |

### **4.3 测试工具与流程**

工具：Postman（接口测试，或者ApiPost）、JMeter（性能测试）、Selenium（UI自动化）、SonarQube（代码质量扫描）。

流程：

单元测试 → 集成测试 → 系统测试 → UAT用户验收测试

**5.部署与维护**

**5.1部署架构**

| **组件** | **部署方案** |
| --- | --- |
| **前端服务** | **Nginx反向代理静态资源（Vue打包文件），支持HTTPS与CDN加速。** |
| **后端服务** | **Spring Boot Jar包部署于Docker容器（2副本+负载均衡），内嵌Tomcat。** |
| **数据库** | **- MySQL（主从架构，阿里云RDS） - MongoDB（分片集群，存储会话历史及非结构化数据）。** |
| **文件存储** | **阿里云OSS（生产环境） / 本地存储（测试环境，Nginx代理路径映射）。** |
| **监控与日志** | |  | | --- | | **Prometheus + Grafana（资源监控），ELK（日志聚合），企业微信告警通知。** | |

**5.2部署流程**

1.环境准备：

通过Docker Compose编排MySQL + Spring Boot + Nginx容器。

初始化数据库表结构（Flyway脚本自动化执行）。

2.服务启动：

# 启动Docker集群

docker-compose -f prod.yml up -d

3.配置校验：

检查Spring Boot Actuator健康接口（/actuator/health）。

验证OSS文件上传权限及MongoDB连接池状态。

**5.3运维计划**

日常维护：

每日定时备份数据库（Binlog增量+全量快照）。

每周清理过期日志（保留周期≤90天）。

故障处理：

预设降级策略：模型API调用失败时自动切换备用服务（如本地缓存响应）。

应急预案：数据库主从切换耗时≤5分钟，服务宕机恢复SLA≥99.9%。

版本迭代：

基于GitLab CI/CD实现自动化构建与灰度发布。

**6.进度安排**

### **6.1进度总安排**

| **周次** | **阶段** | **目标内容** |
| --- | --- | --- |
| 第1周 | 原型设计 | 交互稿+页面流程设计评审 |
| 第1周 | 前端开发 | 登录页、主界面、对话输入区开发 |
| 第2周 | 后端开发 | 模型API封装、用户角色系统、内容存储 |
| 第3周 | 敏感词与日志模块 | 接入违禁词过滤与操作审计功能 |
| 第4周 | 内测优化 | BUG修复、UI优化、系统稳定性调整 |
| 第4周 | 正式上线 | 数据清洗、服务器部署、用户培训 |