

# 企业级大模型克隆平台

一个完整的企业级大模型API接口管理平台，为企业和开发者提供统一的大模型服务接口。

## 核心特性

### API接口服务

- 兼容OpenAI格式的RESTful API接口
- 支持流式和非流式响应 (Server-Sent Events)
- WebSocket实时通信接口
- 支持多种模型调用 (GPT、Deepseek、Claude等)
- 统一的请求/响应格式

### 模型管理系统

- 多模型接入管理 (GPT、Deepseek、Claude等)
- 动态模型切换和负载均衡
- 模型配置管理 (API密钥、限额等)
- 模型健康监控和故障转移

### 用户认证与权限

- JWT身份验证系统
- API密钥管理 (生成、禁用、轮换)
- 用户角色与权限控制
- API访问频率限制



## 管理后台界面

- 用户管理界面（查看、创建、编辑用户）
- 模型配置管理界面
- API使用统计与监控仪表盘
- 系统日志查看
- 实时监控指标展示



## 高可用性功能

- 请求缓存机制
- 异步任务队列处理
- 详细的访问日志记录
- 错误重试和回退机制
- 系统健康检查



## 监控与日志

- API调用统计分析
- 模型使用情况统计
- 性能指标监控（响应时间、成功率等）
- 用户使用行为分析
- 详细的操作日志



## 技术架构

### 后端技术栈

- **框架:** FastAPI (Python 3.9+)
- **数据库:** SQLite (开发) / PostgreSQL (生产)

- **认证:** JWT + API Key
- **缓存:** Redis (可选)
- **任务队列:** Celery (可选)
- **容器化:** Docker + Docker Compose

## 前端技术栈

- **框架:** Vue 3 + TypeScript
- **UI组件:** Element Plus
- **状态管理:** Pinia
- **路由:** Vue Router
- **构建工具:** Vite
- **样式:** CSS Variables + TailwindCSS

## 基础设施

- **反向代理:** Nginx
- **容器编排:** Docker Compose
- **部署:** Docker容器化部署
- **监控:** 内置健康检查



## 快速开始

## 系统要求


- Docker 20.0+
- Docker Compose 2.0+
- 可用端口: 3000 (前端), 8000 (后端)

# 一键启动

```
# 克隆项目
git clone <repository-url>
cd llm-platform

# 启动平台
./scripts/start.sh
```

启动完成后:

-  前端界面: <http://localhost:3000>
-  后端API: <http://localhost:8000>
-  API文档: <http://localhost:8000/docs>

# 默认账户

角色	用户名	密码	权限
管理员	admin	admin123	完整管理权限
演示用户	demo_user	demo123	基础使用权限
开发人员	developer	dev123	基础使用权限

# 使用指南

## 1. 用户管理

### 登录系统

1. 访问 <http://localhost:3000>
2. 使用默认管理员账号登录
3. 进入管理后台

## 创建用户

1. 进入 "管理后台" → "用户管理"
2. 点击 "创建用户" 按钮
3. 填写用户信息并分配角色
4. 保存用户

## 2. API密钥管理

### 生成API密钥

1. 进入 "API密钥" 页面
2. 点击 "生成新密钥" 按钮
3. 设置密钥名称和权限
4. 复制生成的密钥（仅显示一次）

### 使用API密钥

```
curl -X POST "http://localhost:8000/api/v1/chat/completions" \  
  -H "Authorization: Bearer YOUR_API_KEY" \  
  -H "Content-Type: application/json" \  
  -d '{  
    "model": "gpt-4o",  
    "messages": [  
      {"role": "user", "content": "Hello, world!"}  
    ]  
  }'
```

## 3. 模型配置

### 添加新模型

1. 进入 "管理后台" → "模型管理"

2. 点击 "添加模型" 按钮
3. 填写模型信息:
  - 模型名称 (如: gpt-4o)
  - 提供商 (如: openai)
  - 接口地址
  - 模型配置参数
4. 保存配置

## 测试模型

1. 在模型列表中点击 "测试" 按钮
2. 查看测试结果和响应时间
3. 确认模型状态正常

## 4. API接口使用

### 聊天完成接口

```
import requests

url = "http://localhost:8000/api/v1/chat/completions"
headers = {
    "Authorization": "Bearer YOUR_API_KEY",
    "Content-Type": "application/json"
}
data = {
    "model": "gpt-4o",
    "messages": [
        {"role": "user", "content": "写一个Python函数计算斐波那契数列"}
    ],
    "temperature": 0.7,
    "max_tokens": 1000
}

response = requests.post(url, headers=headers, json=data)
print(response.json())
```

## 流式响应

```
import requests

url = "http://localhost:8000/api/v1/chat/completions"
headers = {
    "Authorization": "Bearer YOUR_API_KEY",
    "Content-Type": "application/json"
}
data = {
    "model": "gpt-4o",
    "messages": [
        {"role": "user", "content": "介绍一下人工智能的发展历程"}
    ],
    "stream": True
}

response = requests.post(url, headers=headers, json=data,
stream=True)

for line in response.iter_lines():
    if line:
        print(line.decode('utf-8'))
```

## 高级配置

### 环境变量配置

编辑 `backend/.env` 文件:



```
# 数据库配置
DATABASE_URL=sqlite:///./llm_platform.db

# 安全配置
SECRET_KEY=your-secret-key-change-this-in-production
ACCESS_TOKEN_EXPIRE_MINUTES=1440

# API配置
API_V1_STR=/api/v1
PROJECT_NAME=企业级大模型克隆平台

# CORS配置
BACKEND_CORS_ORIGINS=["http://localhost:3000"]

# 模型提供商密钥（可选）
OPENAI_API_KEY=sk-...
DEEPSEEK_API_KEY=sk-...
CLAUDE_API_KEY=sk-...

# 模拟服务（开发演示用）
USE MOCK_SERVICE=true
```

## PostgreSQL数据库配置

1. 启用PostgreSQL服务（编辑 `docker-compose.yml`）：

```
postgres:
  image: postgres:15-alpine
  environment:
    POSTGRES_DB: llm_platform
    POSTGRES_USER: llm_user
    POSTGRES_PASSWORD: llm_password
  ports:
    - "5432:5432"
```

### 1. 更新数据库连接:

```
DATABASE_URL=postgresql://llm_user:llm_password@postgres:5432/
llm_platform
```

## Redis缓存配置

启用Redis服务:

```
redis:
  image: redis:7-alpine
  ports:
    - "6379:6379"
```



## 监控与运维

### 查看服务状态

```
# 查看所有服务状态
docker-compose ps

# 查看服务日志
docker-compose logs -f

# 查看特定服务日志
docker-compose logs -f backend
docker-compose logs -f frontend
```

### 健康检查

```
# 后端健康检查
curl http://localhost:8000/health

# 前端健康检查
curl http://localhost:3000/health
```

### 性能监控

平台内置了以下监控功能:

- API响应时间统计
- 错误率统计
- 用户使用情况分析
- 模型调用统计
- 系统资源使用情况

访问管理后台查看详细监控数据。

# 开发指南

## 本地开发环境

### 后端开发

```
cd backend

# 创建虚拟环境
python -m venv venv
source venv/bin/activate # Windows: venv\Scripts\activate

# 安装依赖
pip install -r requirements.txt

# 启动开发服务器
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

### 前端开发

```
cd frontend

# 安装依赖
npm install

# 启动开发服务器
npm run dev
```

## 代码结构

```
llm-platform/
├── backend/                                # 后端代码
│   ├── app/
│   │   ├── api/                          # API路由
│   │   ├── core/                        # 核心配置
│   │   ├── crud/                        # 数据库操作
│   │   ├── db/                          # 数据库配置
│   │   ├── middleware/                  # 中间件
│   │   ├── models/                     # 数据模型
│   │   ├── schemas/                    # Pydantic模式
│   │   ├── services/                   # 业务服务
│   │   └── main.py                      # 应用入口
│   ├── scripts/                         # 脚本文件
│   └── requirements.txt                  # Python依赖
├── frontend/                             # 前端代码
│   ├── src/
│   │   ├── api/                        # API接口
│   │   ├── components/                 # Vue组件
│   │   ├── router/                     # 路由配置
│   │   ├── store/                      # 状态管理
│   │   ├── types/                      # TypeScript类型
│   │   ├── utils/                      # 工具函数
│   │   └── views/                      # 页面组件
│   └── package.json                    # Node依赖
├── scripts/                             # 运维脚本
├── docs/                                # 文档
└── docker-compose.yml                   # 容器编排
```

# 故障排除

## 常见问题

### 1. 端口被占用

```
# 查看端口占用
lsof -i :3000
lsof -i :8000

# 杀死占用进程
kill -9 <PID>
```

### 2. Docker容器启动失败

```
# 查看容器日志
docker-compose logs backend
docker-compose logs frontend

# 重新构建容器
docker-compose up --build
```

### 3. 数据库连接失败

```
# 检查数据库文件权限
ls -la backend/data/

# 重新初始化数据库
docker-compose exec backend python scripts/init_demo_data.py
```

## 4. API调用失败

- 检查API密钥是否正确
- 确认请求格式符合OpenAI标准
- 查看后端日志获取详细错误信息

## 日志位置

- 后端日志: `backend/logs/`
- 前端日志: 浏览器控制台
- Docker日志: `docker-compose logs`



## 部署指南

### 生产环境部署

1. 配置生产环境变量:

```
# 使用强密钥
SECRET_KEY=your-super-secret-production-key

# 使用PostgreSQL
DATABASE_URL=postgresql://user:password@localhost:5432/llm_platform

# 配置Redis
REDIS_URL=redis://localhost:6379/0

# 禁用模拟服务
USE MOCK_SERVICE=false

# 配置真实API密钥
OPENAI_API_KEY=sk-...
DEEPSEEK_API_KEY=sk-...
```

## 1. 配置反向代理:

```
server {
    listen 80;
    server_name your-domain.com;

    location / {
        proxy_pass http://localhost:3000;
    }

    location /api/ {
        proxy_pass http://localhost:8000;
    }
}
```

## 1. 启动生产环境:



```
# 使用生产配置启动
docker-compose -f docker-compose.prod.yml up -d
```

## 容器化部署

项目已完全容器化，可直接部署到:

- Docker Swarm
- Kubernetes
- 云容器服务 (AWS ECS, 阿里云ACK等)

## 贡献指南

欢迎提交Issue和Pull Request!

## 开发流程

1. Fork项目
2. 创建功能分支
3. 提交代码
4. 创建Pull Request

## 代码规范

- 后端: 遵循PEP 8
- 前端: 遵循Vue官方风格指南
- 提交信息: 使用约定式提交格式

## 许可证

本项目采用 MIT 许可证 - 查看 [LICENSE](#) 文件了解详情。



如有问题，请：

1. 查看本文档的故障排除部分
2. 搜索现有的Issue
3. 创建新的Issue描述问题

---

 **立即体验企业级大模型克隆平台！**

```
./scripts/start.sh
```

访问 <http://localhost:3000> 开始您的AI之旅！