

**项目架构设计文档（B）**

成员：宋子晗（前端开发工程师）

刘浩（后端开发工程师）

**设计小组：aA冠军队**

**AI虚拟角色聊天平台**

**架构设计文档**

**aA冠军队**

**1. 引言**

文档目的

本文档描述了系统的架构设计，包括系统的整体结构、模块划分、各模块间的交互以及关键技术选型。目标是为开发人员、测试人员和其他相关人员提供详细的技术指导。

适用范围

本文档适用于系统开发过程中的设计阶段，尤其是在开发、集成和部署阶段。

定义、缩略语和缩写

API：应用程序编程接口

DB：数据库

UI：用户界面

REST：表述性状态转移

**2. 系统概述**

**2.1 系统目标**

该系统旨在为用户提供一个高智能化、高交互性的客服平台，通过自然语言处理（NLP）与语音识别技术，实现人与虚拟角色的高效沟通。系统不仅支持多种角色选择与个性化交互，还能够实现文本与语音双向输入输出，提供实时聊天、消息记录以及多设备同步等功能。同时，系统注重用户体验和响应速度，确保在大并发环境下依然保持高可用、低延迟的服务性能，为企业或个人用户提供可靠的智能客服解决方案。

**2.2 系统功能**

系统最终实现的功能包括但不限于以下几类：

用户登录与身份管理：

1. 提供安全的用户注册、登录、注销功能。
2. 支持多种身份验证方式（用户名/邮箱 + 密码、第三方 OAuth 登录等）。
3. 用户资料管理，包括头像、昵称、个人偏好设置等。

角色选择与切换：

1. 提供多种虚拟角色选择，每个角色具有不同的交互风格和背景设定。
2. 支持用户在聊天过程中实时切换角色，角色切换后聊天上下文可选择保留或清空。
3. 每个角色可配置头像、背景、语音风格等个性化信息。

文本与语音输入：

1. 用户可以通过文本输入发送消息，系统实时响应。
2. 支持语音输入，系统通过语音识别将语音转换为文本，并生成回复。
3. 支持语音输出，系统可以将回复内容转换为语音播放，提供沉浸式体验。

消息记录与实时聊天：

1. 支持实时聊天，确保用户与虚拟角色的交互即时响应。

前后端交互与服务支持：

1. 前端通过 REST API 或 WebSocket 与后端交互，实现消息的实时传输。
2. 后端提供高并发处理能力、消息队列管理、音频流处理等服务。

**2.3 系统约束**

系统在设计和实现过程中需要遵循以下约束条件：

浏览器兼容性：

系统前端支持主流浏览器：Chrome、Firefox、Edge，保证跨平台用户体验。

部署环境

系统部署在

安全性与数据保护

所有数据传输采用 HTTPS 加密，确保通信安全。

用户数据加密存储，并通过认证和授权机制保护敏感信息。

支持访问控制策略，限制用户不同角色对系统功能的权限。

性能与可靠性

系统需满足高并发访问场景下的稳定性要求。

语音与文本交互需保证低延迟 (<300ms)，提供流畅的用户体验。

1. **架构设计**

**3.1 架构图**

描述：

本系统采用 典型的前后端分离架构，同时支持文本与语音的实时交互。前端使用 Vue.js 构建界面，并通过 WebSocket 与后端进行实时通信。后端由 Node.js/Express 提供 API 服务，负责处理用户请求、管理会话、处理语音流。音频流通过 WebRTC 或 WebSocket 实时传输，实现低延迟语音交互。数据库使用 MongoDB 存储用户数据、角色信息和聊天记录。

**3.2 系统组件说明**

3.2.1 前端

技术栈： Vue.js、Vuex、Element Plus、Axios、WebSocket

功能：

负责用户界面展示与交互。

提供多角色选择和切换界面。

支持文本消息输入和语音输入。

使用 WebSocket 或 WebRTC 实时接收后端消息和音频流。

消息列表自动滚动、加载动画和状态提示（正在输入/思考）。

3.2.2 后端

技术栈： Node.js、Express、WebSocket、WebRTC

功能：

提供 REST API 服务处理用户请求。

管理聊天会话和消息队列，实现会话的持久化和多用户并发。

调用语音识别 (STT) 和语音合成 (TTS) 服务，实现语音聊天功能。

支持 AI 角色对话生成，包括上下文管理和多角色个性化回复。

提供安全认证和授权接口，保证用户数据安全。

3.2.3 数据库

技术栈： MongoDB

功能：

存储用户账户信息、角色配置和聊天历史。

提供高性能查询和扩展能力，支持多会话和多角色管理。

可与 Redis 缓存结合，存储活跃会话状态，提高响应速度。

3.2.4 音频处理

技术栈： WebRTC、WebSocket、第三方语音 API（如百度/阿里/Tencent TTS/STT）

功能：

实现前端与后端之间的低延迟音频流传输。

通过 STT 将语音转换为文本输入给 AI 角色处理。

将 AI 角色文本回复通过 TTS 转换为语音播放给用户。

支持实时音量检测、录音状态控制和语音播放队列管理。

**3.3 系统特点**

前后端分离：前端专注界面交互，后端处理业务逻辑和数据存储。

实时交互：WebSocket 和 WebRTC 实现低延迟消息和音频传输。

多角色支持：用户可选择不同角色与 AI 进行对话，角色个性化配置可独立加载。

语音与文本融合：同时支持文本聊天和语音聊天，实现多模态交互体验。

高可扩展性：MongoDB 和 WebSocket 架构支持多用户并发，系统易于横向扩展。

1. **模块设计**

**4.1 前端模块设计**

模块1：用户管理

功能描述：负责用户的注册、登录、身份验证。

接口：

POST /user/login：用户登录接口

POST /user/register：用户注册接口

模块2：聊天主界面—左边栏

功能描述：实现AI角色的选择、头像与信息的显示

接口：

POST /api/assistant/list 获取角色信息

模块3：聊天主界面—信息框

功能描述：获取用户与AI传入的信息并显示在界面上

POST /api/chat：上传用户消息并接收后端返回消息

GET /api/voice/{id}：获取语音消息

**4.2 后端模块设计**

技术实现：使用 WebSocket 进行音频流的实时传输，后端使用 WebRTC 进行音频数据的处理。

**5. 技术选型**

**5.1前端技术**

Vue.js：用于构建单页面应用，支持响应式设计。

Element Plus：用于构建 UI，提供丰富的组件和样式。

Axios：用于发送 HTTP 请求，获取数据。

WebSocket：用于实现实时的消息传输与音频流传输。

**5.2后端技术**

Node.js：高性能的 JavaScript 运行环境，适合处理高并发的请求。

Express：用于构建 RESTful API 服务。

MongoDB：NoSQL 数据库，适用于存储用户信息、消息数据等。

音频处理技术

WebRTC：用于实现实时语音通讯，支持音频流的传输和处理。

WebSocket：用于实时传输音频数据流。

**6. 数据流与交互**

消息流程

用户发送文本消息：前端通过 POST /api/chat 接口发送消息，后端处理后存储并推送给对方用户。

用户发送语音消息：前端通过录音设备获取音频，上传至后端，后端保存并返回音频流。

WebSocket 流程

用户连接 WebSocket 服务后，可以接收到实时的消息和音频数据。

后端将生成的音频流通过 WebSocket 传输至前端，前端实时播放。

**7. 安全设计**

身份认证：使用 JWT（JSON Web Token）进行用户认证，所有敏感接口需要带上 Authorization 头。

数据加密：通过 HTTPS 协议加密前后端数据传输，确保传输过程中的数据安全。

**8. 性能与扩展性**

性能优化

前端使用懒加载与按需加载技术，减少初次加载时间。

后端使用缓存技术（如 Redis）提高接口响应速度。

扩展性：

后端可以通过增加服务节点进行横向扩展，提升并发处理能力。

数据库采用分库分表策略以支持大规模用户数据存储。

**9. 测试计划**

单元测试：对系统的每个模块进行单元测试，确保基本功能的正确性。

集成测试：测试系统中各个模块之间的交互和数据流。

性能测试：对系统进行负载测试和压力测试，确保在高并发情况下系统的稳定性。

**10. 部署与运维**

部署环境

前端：通过 Nginx 部署静态文件，支持 CDN 加速。

后端：通过 Docker 容器化部署，使用 Kubernetes 进行自动化管理。

数据库：使用 MongoDB Atlas 提供的云数据库服务。

运维方案

监控：使用 Prometheus 和 Grafana 监控系统健康状态。

日志管理：使用 ELK（Elasticsearch、Logstash、Kibana）进行日志的收集与分析。

**11. 总结**

本系统的架构设计考虑了可扩展性、高可用性和安全性，通过分层架构和模块化设计确保系统的维护性与可扩展性。通过合理的技术选型与方案设计，能够实现高效、实时的消息和音频流传输，满足用户需求。