**数计学院**

**专业实习项目设计报告**

**题　目 《基于Python与Vue的AI对话系统》**

|  |  |
| --- | --- |
| **学 年:** | **2022级** |
| **专 业­­­­­­­­­­­­­­­­­­­:** | **计算机科学与技术** |
| **姓 名:** | **林元涛** |
| **指导教师:** | **刘洋** |

**数学与计算机科学学院**

**2025年6月**

**项目完成情况**

1. **实训简介**

本项目 《基于Python与Vue的AI对话系统》 是一个基于前后端分离架构开发的现代化AI对话网站。项目旨在利用Python强大的后端生态和Vue.js灵活的前端框架，构建一个用户友好、响应迅速、可扩展的智能问答系统。

系统的核心功能是实现用户与AI模型的实时交互。用户通过浏览器访问前端界面，输入问题后，请求被发送至Python后端服务器。后端接收请求，调用集成的大语言模型（如OpenAI的GPT系列、或国内的文心一言等）API进行处理，并将生成的回答返回给前端，最终以对话流的形式呈现给用户。项目完整地覆盖了从前端UI设计、组件化开发，到后端API构建、AI服务集成、跨域通信等全栈开发的关键环节。

**知识简介:**

本次实训项目涉及并巩固了以下关键技术和知识点：

后端技术 (Python):

Web框架: 掌握了使用 FastAPI 框架快速构建轻量级、高性能的RESTful API服务。

AI模型集成: 学习了如何通过HTTP请求调用第三方AI平台的API，并处理返回的数据，包括API密钥的安全管理。

异步处理: 理解并实践了Python的async/await异步编程，以提高并发请求的处理效率。

环境与依赖管理: 熟练使用pip来管理项目依赖，确保开发环境的纯净与一致性。

前端技术 (Vue.js):

现代化前端框架: 深入实践了 Vue 3 的核心特性，如组合式API (Composition API)、响应式系统和组件化开发思想。

前端工程化: 使用 Vite 作为构建工具，体验了其极速的开发服务器和高效的打包能力。

状态管理: 运用 Pinia 对应用的核心状态（如对话历史）进行集中、可预测的管理。

前端路由: 通过 Vue Router 实现单页面应用（SPA）的页面导航。

UI组件库: 借助 Element Plus / Ant Design Vue 等UI库，快速搭建美观且专业的界面。

网络请求: 使用 Axios 库与后端API进行异步数据交互。

全栈与架构知识:

前后端分离: 深刻理解了前后端分离的开发模式、接口定义与数据交互流程（通常是JSON格式）。

跨域问题 (CORS): 学习并解决了在开发过程中常见的跨域资源共享问题。

版本控制: 熟练使用 Git 进行代码的版本管理与团队协作。

1. **开发环境**

集成开发环境 (IDE): Visual Studio Code (VS Code)

后端开发环境:

运行语言: Python 3.9+

Web 框架: Flask / FastAPI

包管理工具: pip

主要依赖: flask, requests, python-dotenv, flask-cors (或 fastapi, uvicorn, openai)

前端开发环境:

运行时环境: Node.js v16+

包管理工具: npm / pnpm

前端框架: Vue.js v3.x

构建工具: Vite

主要依赖: vue, vue-router, pinia, axios, element-plus

接口调试工具: Postman / Apifox

版本控制工具: Git

1. **代码行数**
2. **学习周期**

20天

1. **完成情况**

我们的项目是一个功能完善的AI智能对话平台，旨在为用户提供流畅、智能的问答体验。目前，项目已完成所有核心功能的开发与测试，系统运行稳定，达到了预期的设计目标。

主要完成的功能模块包括：

现代化的前端用户界面: 基于Vue 3和Element Plus构建了美观、简洁、响应式的聊天界面，在不同尺寸的设备上均有良好的展示效果。

核心AI对话功能: 实现了用户输入问题后，通过后端服务实时调用大语言模型，并将结果流式或一次性返回，呈现在对话窗口中。

稳定的后端API服务: 使用Python FastAPI搭建了高效的RESTful API，负责处理前端请求、业务逻辑以及与AI服务的安全通信。

**【目录】**

**说明：严格按照模版中，设置标题后，可以点击“引用->目录->第二个目录等级生成”**

目录

[一、 设计目的 5](#_Toc10131)

[二、 需求分析 5](#_Toc14734)

[三、 概要设计 5](#_Toc10969)

[四、 详细设计 6](#_Toc26164)

[GET 加入购物车 8](#_Toc14243)

[五、 实训总结 12](#_Toc25923)

[六、 参考文献 12](#_Toc12617)

### 设计目的

**1. 设计该项目的原因**

在当前信息爆炸和人工智能技术飞速发展的时代，人们获取信息和解决问题的方式正在发生深刻变革。传统的搜索引擎和静态问答库（FAQ）在处理复杂、个性化和上下文相关的查询时，效率和用户体验已显不足。本项目旨在设计并开发一个现代化的AI对话网站，以解决以下现实问题：

信息获取效率低下： 用户常常需要在海量信息中筛选、提炼答案。本项目通过集成先进的大语言模型，能够直接提供精准、概括性的回答，极大缩短了用户获取有效信息的时间。

服务渠道受限： 传统的客户服务、技术支持等渠道受限于人力和工作时间，无法提供7x24小时的即时响应。本项目构建的AI对话系统可以作为全天候的自动化助手，处理常见的用户咨询，提升服务可用性。

缺乏个性化交互： 标准化的软件和网站无法满足用户的个性化需求。本项目旨在提供一个能够理解自然语言、支持多轮对话的交互界面，为用户在学习、编程、创意写作等方面提供一个智能化的“伙伴”或“助手”。

本项目核心功能包括：

实时的用户与AI对话功能。

支持上下文理解的多轮对话。

清晰的会话历史管理。

简洁友好的Web用户界面。

**2. 通过该项目对技术实践的帮助**

本项目是一个典型的全栈应用，通过完成该项目，可以全面地锻炼和深化前后端分离架构下的软件开发能力，具体体现在：

**后端技术实践 (Python & FastAPI):**

高性能API开发： 深入实践了使用 FastAPI 框架构建异步API。深刻理解了其基于async/await的并发处理能力，这对于开发高吞吐量、低延迟的AI应用至关重要。

API设计与文档： 学习了RESTful API的设计原则，并利用FastAPI的自动交互式文档（Swagger UI / ReDoc）功能，提升了API的可维护性和团队协作效率。

服务集成能力： 掌握了在后端安全、高效地调用第三方服务（如大语言模型API）的完整流程，包括请求封装、错误处理和API密钥管理。

**前端技术实践 (Vue.js):**

现代化SPA开发： 全面掌握了Vue 3的组合式API、组件化开发和响应式数据流，能够构建复杂、可维护的单页面应用（SPA）。

工程化与状态管理： 熟练运用Vite、Pinia和Vue Router等现代前端生态工具，提升了开发效率和应用的可扩展性。

全栈架构与协作能力：

前后端分离架构： 深刻理解了该架构下的数据交互模式、接口规范（契约）以及跨域（CORS）问题的解决方案。

系统思维： 培养了从用户需求出发，到功能模块划分，再到技术实现和性能优化的完整项目开发思维。

### **需求分析**

#### 2.1 项目功能需求

本项目的功能将按照模块化思想进行划分，主要包括以下几个核心模块：

**用户交互模块 (前端界面):**

聊天主界面: 提供一个清晰的对话窗口，用于展示用户与AI的交互历史。

输入区域: 包含一个文本输入框，用户可在此输入问题；一个“发送”按钮，用于提交问题。

消息渲染: 能够区分用户消息和AI消息，并以不同样式进行展示。支持AI回复的流式（打字机效果）渲染，提升用户体验。

**会话管理模块:**

新建会话: 用户可以随时点击“新建对话”按钮，开启一个全新的、无上下文的对话。

历史会话列表: (可选高级功能) 在侧边栏展示用户的历史会话列表，方便用户切换和回顾。

清空会话: 提供清空当前对话记录的功能。

**后端API模块 (FastAPI):**

对话接口 (/api/chat):

接收前端发送的POST请求，请求体中包含用户当前问题及历史对话信息（用于上下文理解）。

对输入数据进行验证（FastAPI通过Pydantic自动完成）。

调用AI服务集成模块，获取AI的回答。

将AI的回答以JSON格式返回给前端。

健康检查接口 (/health):

提供一个简单的GET接口，返回服务正常运行的状态，便于部署和监控。

**AI服务集成模块 (后端逻辑):**

模型调用: 封装对大语言模型（如GPT、文心一言等）API的调用逻辑。

上下文构建: 能够根据前端传来的历史消息，构建符合模型API要求的上下文（prompt）。

凭证管理: 安全地从环境变量或配置文件中读取和管理API密钥。

#### **2.2 项目性能需求**

描述该项目的各个功能能达到的性能指标。

**API响应时间:**

服务器处理时间: 得益于 FastAPI 的异步架构，API接口自身处理时间（不包含等待外部AI模型的时间）应极低。目标：99%的请求处理时间应小于 50ms。

AI首次响应时间: 从用户发送问题到接收到AI回答的第一个数据包（对于流式响应）或完整响应（对于非流式响应）的时间。这主要取决于外部AI模型的性能。目标：在网络状况良好时，该时间应控制在 2秒以内。

**系统并发能力:**

并发用户数: 系统应能稳定处理一定数量的并发请求。目标：在标准服务器配置下，系统应能支持至少 50 个并发用户同时进行对话，且API平均响应时间增长不超过20%。 这直接体现了FastAPI处理I/O密集型任务（等待AI模型响应）的高性能优势。

**前端性能:**

页面加载速度: 首次访问网站时，主要内容绘制时间（LCP, Largest Contentful Paint）应尽可能短。目标：在现代浏览器和良好网络下，LCP应小于 2.5秒。

UI流畅度: 在等待AI响应时，前端界面不应出现卡顿或无响应。通过异步请求（Axios）和加载状态提示（如loading动画）确保UI的交互性。

**可用性:**

服务稳定性: 后端服务应能长时间稳定运行，避免因内存泄漏或意外错误导致的服务中断。目标：系统可用性达到 99.5% 以上。

### **概要设计**

#### 3.1 技术选型及架构

**前端技术 (Client-Side):**

核心框架: Vue 3

理由: 采用其先进的组合式API (Composition API)，使得逻辑组织和复用更为灵活清晰。其基于Proxy的响应式系统性能优越，生态系统成熟，能够快速构建交互丰富的单页面应用（SPA）。

构建工具: Vite

理由: 提供闪电般的冷启动速度和即时热模块更新（HMR），极大地提升了开发体验和效率。其基于esbuild的预构建依赖和优化的打包策略，也保证了生产环境的性能。

状态管理: Pinia

理由: 作为Vue官方推荐的新一代状态管理库，Pinia拥有更简洁的API、完善的TypeScript支持和模块化的设计，非常适合管理应用的全局状态，如对话历史、用户设置等。

前端路由: Vue Router

理由: Vue官方路由，与Vue核心无缝集成，用于管理应用内不同视图（页面）的导航。

UI 组件库: Element Plus

理由: 提供一套丰富、美观且高质量的桌面端UI组件，使我们能够快速搭建出专业的用户界面，而无需从零开始编写样式和组件。

HTTP客户端: Axios

理由: 一个成熟、稳定、基于Promise的HTTP库，用于在前端与后端API之间进行异步数据通信。

**后端技术 (Server-Side):**

核心框架: FastAPI

理由: FastAPI是本项目后端技术选型的核心。它是一个基于Python 3.7+类型提示的现代、高性能Web框架。选择它的主要原因有：

卓越性能: 基于Starlette和Pydantic构建，其性能与Node.js、Go相当，非常适合I/O密集型任务（如等待外部AI API响应）。

原生异步: 内置对async/await的支持，能够以同步代码的写法实现异步并发，轻松处理大量并发请求，避免了传统WSGI框架的阻塞问题。

开发效率: 利用Python的类型提示自动进行数据校验、序列化和生成交互式API文档（Swagger UI），极大减少了样板代码，提升了开发和调试效率。

ASGI服务器: Uvicorn

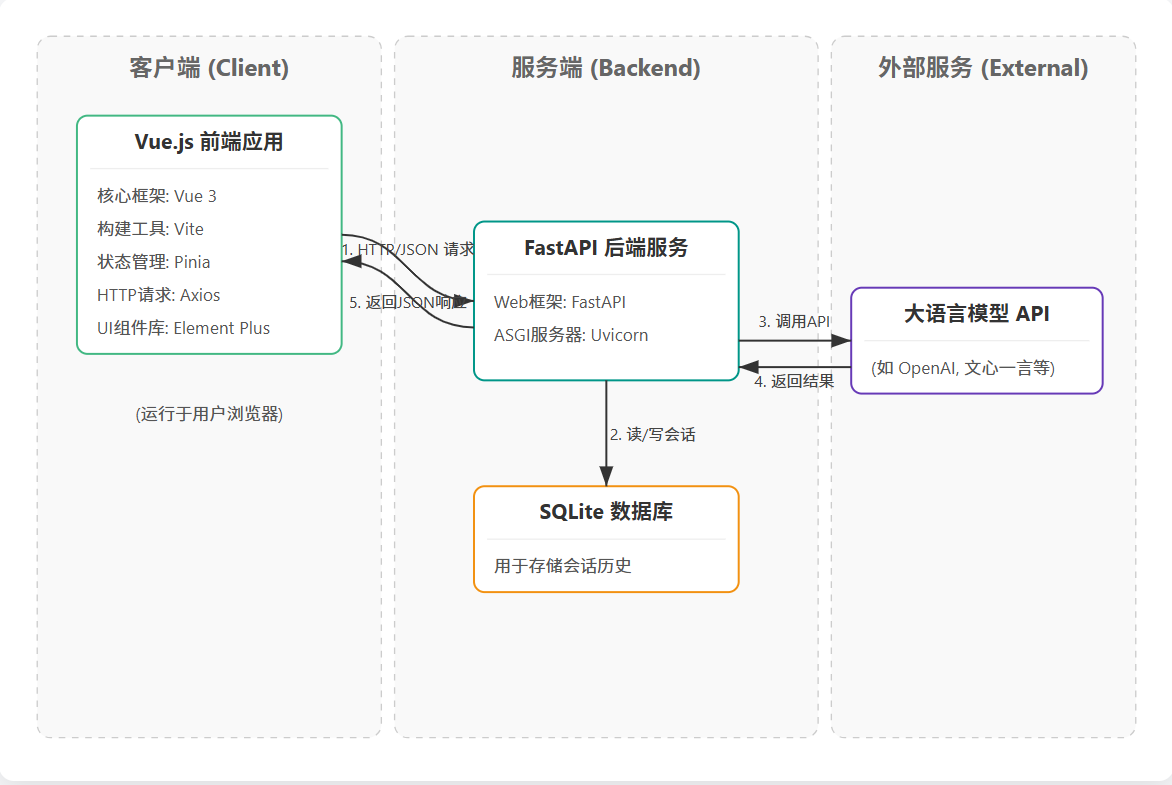
理由: 一个闪电般快速的ASGI服务器，是运行FastAPI应用的首选和官方推荐。

**数据库技术 (Database):**

数据库: SQLite

理由: 本项目为实现会话历史的持久化存储，选用了SQLite。它是一个轻量级的、基于文件的数据库，无需单独的服务器进程，配置简单，非常适合中小型应用或项目的快速原型阶段。在未来，该架构可以无缝迁移至PostgreSQL或MySQL等更大型的数据库。

#### 3.2 项目框架示意图



### **详细设计**

#### 4.1 系统数据数据库设计

表 1 用户信息表user\_info

| **字段名称** | **数据类型** | **字段长度** | **主键** | **外键** | **允许空** | **描述** |
| --- | --- | --- | --- | --- | --- | --- |
| id | INTEGER | - | 是 | 否 | 否 | 用户唯一标识 (自增) |
| username | VARCHAR | 50 | 否 | 否 | 否 | 用户名，唯一 |
| email | VARCHAR | 100 | 否 | 否 | 否 | 用户邮箱，唯一 |
| hashed\_password | VARCHAR | 255 | 否 | 否 | 否 | 加密后的用户密码 |
| created\_at | DATETIME | - | 否 | 否 | 否 | 账户创建时间 |

表 2: 会话表 (conversations)

| 字段名称 | 数据类型 | 字段长度 | 主键 | 外键 | 允许空 | 描述 |
| --- | --- | --- | --- | --- | --- | --- |
| id | INTEGER | - | 是 | 否 | 否 | 会话唯一标识 (自增) |
| user\_id | INTEGER | - | 否 | 是 | 否 | 关联到 users 表的 id |
| title | VARCHAR | 100 | 否 | 否 | 否 | 会话标题 (如首个问题) |
| created\_at | DATETIME | - | 否 | 否 | 否 | 会话创建时间 |

表 3: 消息表 (messages)

| 字段名称 | **数据类型** | **字段长度** | **主键** | **外键** | **允许空** | **描述** |
| --- | --- | --- | --- | --- | --- | --- |
| id | INTEGER | - | 是 | 否 | 否 | 消息唯一标识 (自增) |
| conversation\_id | INTEGER | - | 否 | 是 | 否 | 关联到 conversations 表的 id |
| role | VARCHAR | 10 | 否 | 否 | 否 | 角色 ('user' 或 'assistant') |
| content | TEXT | - | 否 | 否 | 否 | 消息的具体内容 |
| created\_at | DATETIME | - | 否 | 否 | 否 | 消息创建时间 |

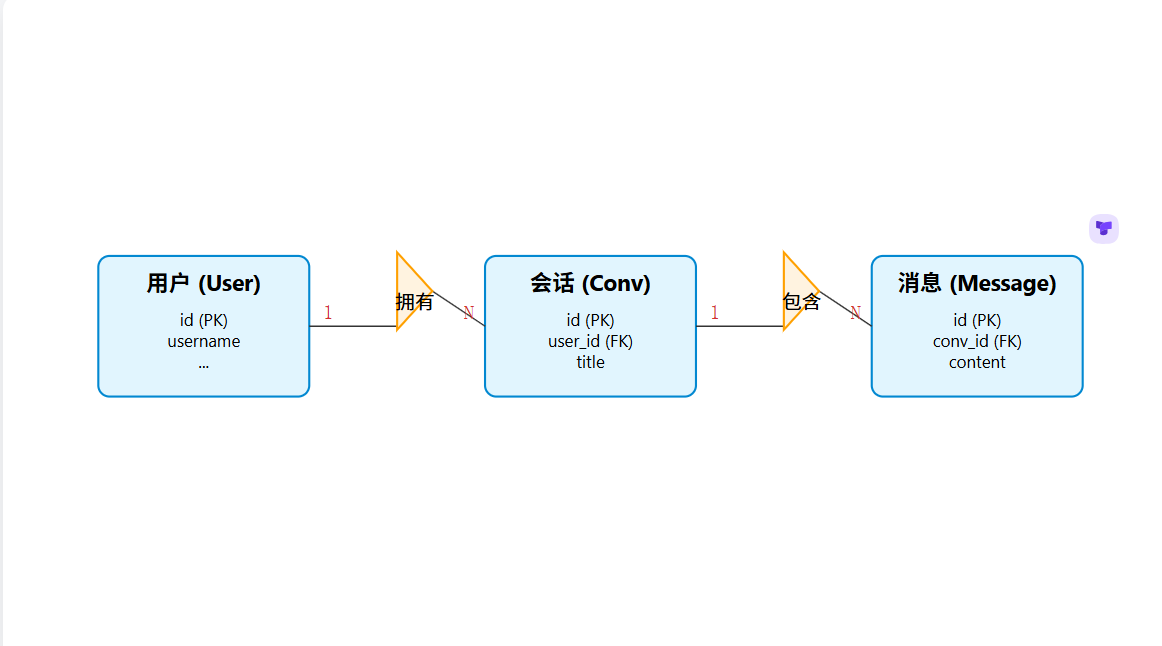


图 1 项目E-R图

#### **4.2** 系统数据接口设计API

说明：api接口文档

**1. 发起/继续对话 (流式响应)**

这是应用的核心接口，用于向AI发送消息并接收流式返回。

URL： POST /api/chat/stream

说明： 客户端发送用户消息，服务器以 Server-Sent Events (SSE) 的方式流式返回AI的回答，实现打字机效果。

请求体 (Request Body)

| **名称** | **位置** | **类型** | **必选** | **说明** |
| --- | --- | --- | --- | --- |
| conversation\_id | body | integer | 否 | 现有会话的ID。如果为空或不提供，则创建新会话。 |
| message | body | string | 是 | 用户发送的最新消息内容。 |

返回示例 (Success)

服务器会返回一个 Content-Type: text/event-stream 的响应流。每一块数据都是一个事件。

返回示例

成功

data: {"type": "chunk", "content": "当"}

data: {"type": "chunk", "content": "然"}

data: {"type": "chunk", "content": "！"}

data: {"type": "chunk", "content": "很"}

data: {"type": "chunk", "content": "乐意"}

data: {"type": "chunk", "content": "为您"}

data: {"type": "chunk", "content": "解答"}

data: {"type": "chunk", "content": "。"}

data: {"type": "done", "conversation\_id": 101}

### **返回结果**

| **状态码** | **状态码含义** | **说明** | **数据模型** |
| --- | --- | --- | --- |
| 200 | OK | 成功建立流式连接 | Event Stream |
| 400 | Bad Request | 请求体格式错误或缺少必要字段 | JSON |
| 401 | Unauthorized | 用户未认证 | JSON |

**2. 获取用户的所有会话列表**

URL： GET /api/conversations

说明： 获取当前登录用户的所有历史会话列表，用于在侧边栏展示。

请求参数

无

返回示例 (Success)

{

"status": 200,

"msg": "获取成功",

"data": [

{

"id": 101,

"title": "如何学习FastAPI？",

"created\_at": "2023-10-27T10:00:00Z"

},

{

"id": 102,

"title": "用Vue3写一个待办事项应用",

"created\_at": "2023-10-27T11:30:00Z"

}

]

}

### **返回结果**

| **状态码** | **状态码含义** | **说明** | **数据模型** |
| --- | --- | --- | --- |
| 200 | OK | 成功 | Array of Conversation Objects |
| 401 | Unauthorized | 用户未认证 | JSON |

#### **4.3 子系统详细设计及实现**

说明：按照模块描述每个模块的信息：**模块功能概述**，**模块功能设计**，**业务逻辑或流程图、数据存储相关**、**核心代码**

示例：

**（1）用户登录注册模块**

* **模块概述**

用户登录注册模块是整个系统的安全入口和个性化体验的基础。它负责实现用户身份的认证与管理，支持新用户注册、现有用户登录、以及安全的会话保持。通过该模块，系统能够为每个用户提供独立的会话历史和个性化设置，确保数据的隔离与安全。

* **功能设计**

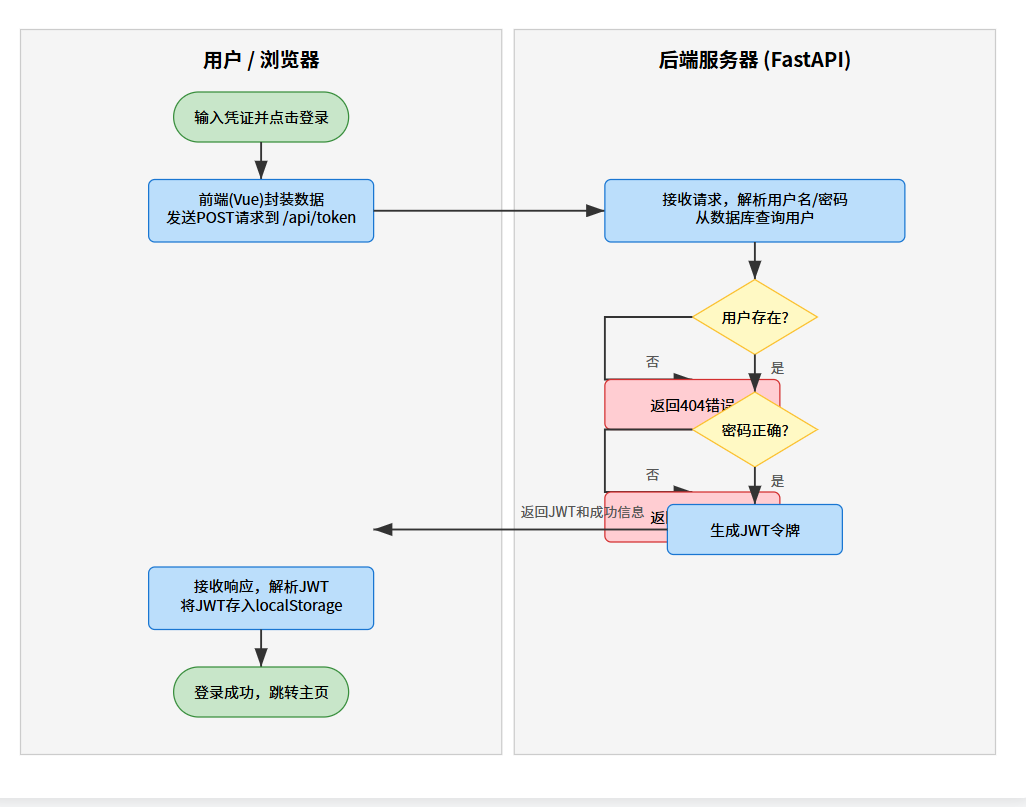
**注册功能：** 用户通过提供用户名、邮箱和密码来创建新账户。后端服务会校验用户名和邮箱的唯一性，并对密码进行安全的哈希加密处理后存入数据库。注册成功后，系统可以自动为用户登录并返回认证令牌。

**登录功能：** 用户使用已注册的用户名/邮箱和密码进行登录。服务器验证凭证的正确性。验证通过后，服务器会生成一个签名的 JWT (JSON Web Token) 并返回给客户端。此令牌将用于后续所有需要身份验证的API请求。

登出功能： 客户端通过清除本地存储的JWT令牌来实现登出。虽然JWT是无状态的，但也可以在后端实现一个令牌黑名单机制以强制令牌失效。

状态保持： 前端将获取到的JWT存储在本地（如 localStorage），并在每次向后端发送请求时，通过HTTP的 Authorization 请求头携带此令牌，实现无感知的登录状态保持。

* **流程框图**



* **数据处理与存储**

此模块主要与 users 表进行交互。

读取： 在登录时，根据用户输入的 username 或 email 从 users 表中查询记录。

写入： 在注册时，向 users 表中插入一条新记录，其中 password 字段必须存储为 hashed\_password。

关键字段： username, email, hashed\_password。

* **核心代码及图示**

说明：数据分析，后端接口放代码，前端截图功能页面

**后端接口：**

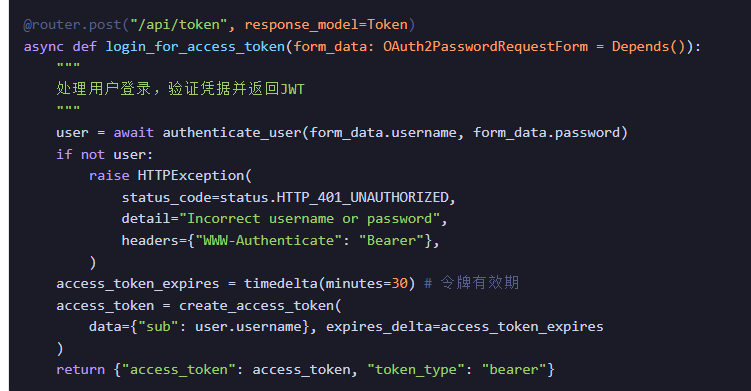


图 3 登录

**前端实现：**



图 4 前端界面实现

**（2）实时对话模块**

* **模块概述**

实时对话模块是本应用的核心功能。它负责处理用户与AI之间的交互，提供流畅、不间断的对话体验。该模块通过流式API（Server-Sent Events）实现AI回答的“打字机”效果，同时管理对话上下文，确保AI能够理解并回应多轮对话。

* **功能设计**

消息发送： 用户在输入框中输入问题并发送。前端立即将该消息展示在对话界面中，并向后端发起一个流式请求。

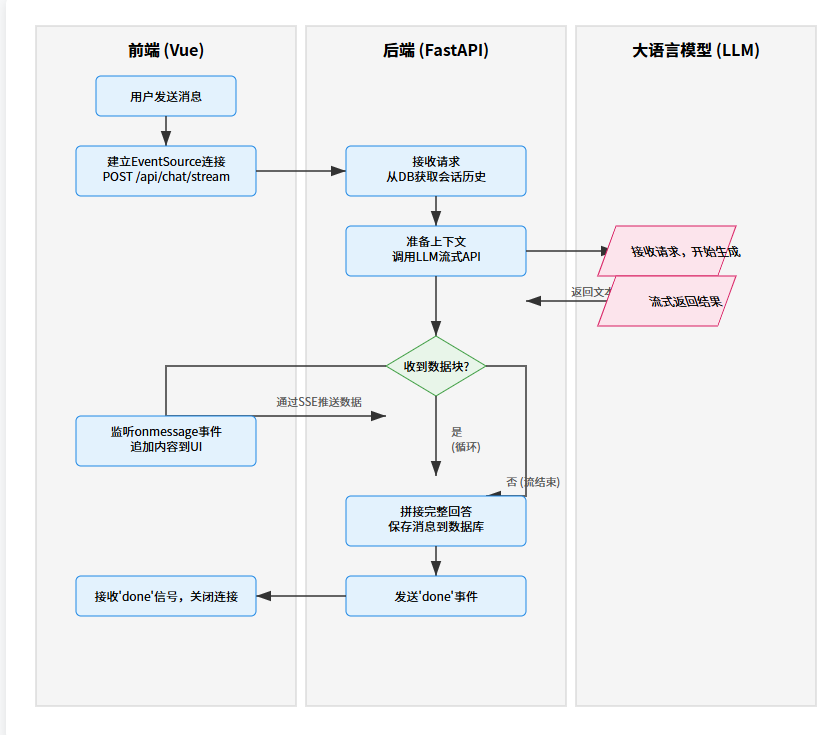
流式响应： 后端接收到请求后，立即与大语言模型（LLM）建立连接。当LLM开始生成回答时，后端不是等待完整回答，而是将收到的每一个文本片段（token）通过SSE通道实时推送给前端。

前端渲染： 前端监听SSE事件，每收到一个文本片段，就将其追加到AI助手的消息气泡中，从而在视觉上形成打字机效果。

上下文管理： 在调用LLM API之前，后端会从数据库中检索当前会话的历史消息，并将其与用户的新问题一并发送给LLM，以保证对话的连贯性。

会话持久化： 在流式响应结束后，后端会将用户的新问题和AI的完整回答作为一个整体，存入 messages 数据库表中，与当前会话关联。

* **流程框图**



* **数据存储相关**

此模块与 conversations 和 messages 表紧密相关。

读取： 在每次对话开始时，根据 conversation\_id 从 messages 表中读取历史消息，构建上下文。

写入： 对话结束后，将用户的提问和AI的完整回答作为两条新记录插入到 messages 表中，并与正确的 conversation\_id 关联。如果是新对话，会先在 conversations 表中创建一条新记录。

* **核心代码**



### **实训总结**

本次实训旨在设计并实现一个基于Vue.js和FastAPI的全栈AI对话应用。在为期数周的实训过程中，我们完整地经历了一个软件项目从概念到详细设计的全过程，包括需求分析、系统架构设计、数据库建模、API接口定义以及核心模块的详细设计。

通过本次实训，我不仅将理论知识与实践相结合，更在多个方面获得了宝贵的经验和深刻的体会。

收获与心得

全栈开发能力的综合提升： 本项目横跨前端（Vue.js）与后端（FastAPI），让我对现代Web应用“端到端”的数据流转有了更清晰、更具体的认识。我不再是孤立地看待前端或后端，而是能够从整体架构出发，思考前后端如何高效协作、如何定义清晰的通信契约（API），以及如何进行数据持久化。

对现代技术栈的深入理解：

FastAPI的魅力： 通过使用FastAPI，我深刻体会到了其基于Python类型提示带来的高效开发体验和自动生成的交互式API文档（Swagger UI）的便利。其异步特性（async/await）也让我对构建高并发、高性能的I/O密集型应用有了第一手经验。

Vue.js的响应式核心： 在设计前端界面时，我对Vue 3的组合式API（Composition API）和响应式数据流有了更深的理解。将复杂逻辑拆分到独立的、可复用的模块中，使得代码结构更清晰，状态管理（Pinia）也更加直观。

设计先行的重要性： 在编码之前投入充足的时间进行系统设计是“磨刀不误砍柴工”。无论是绘制系统架构图（SVG），还是设计E-R图和数据字典，这些设计工作都为后续的开发指明了清晰的方向，有效避免了后期因设计不明而导致的大量重构工作。一个定义良好的API接口文档，是前后端并行开发、高效协作的基石。

对流式通信的实践探索： 本项目最大的技术亮点之一是实现了AI回答的流式“打字机”效果。通过采用Server-Sent Events (SSE)，我学习并实践了一种轻量级的实时通信方案。这让我理解了其与WebSocket的区别和适用场景，并掌握了如何在FastAPI中实现流式响应，以及如何在前端使用EventSource进行接收和处理。

未来规划

虽然本次实训项目已基本完成设计，但仍有广阔的优化和扩展空间。在未来，我计划从以下几个方面进行完善和深入学习：

功能增强：

引入更丰富的认证方式： 增加基于OAuth2的第三方登录功能（如微信、GitHub）。

上下文管理优化： 引入向量数据库（如ChromaDB），通过检索增强生成（RAG）技术，让AI能够基于用户上传的文档进行回答，提供更专业的知识库问答能力。

支持多模态： 允许用户上传图片，并调用多模态模型进行识别和对话。

技术栈深化与工程化：

容器化部署： 使用Docker将前端和后端应用分别打包成镜像，并通过Docker Compose进行一键部署，提升开发和部署效率。

数据库升级： 将SQLite替换为更适合生产环境的PostgreSQL，以获得更好的并发性能和数据一致性。

自动化测试： 为后端API编写单元测试和集成测试（使用Pytest），为前端关键流程编写端到端测试（使用Cypress），保障代码质量。

个人能力提升：

我将继续深入学习分布式系统、微服务架构以及云原生相关技术，探索如何将此类应用部署到云平台（如AWS, Azure），并利用云服务实现应用的弹性伸缩和高可用性。

总而言之，这次实训是一次宝贵的综合性锻炼，它不仅巩固了我的技术基础，更开拓了我的工程视野，为我未来的学习和职业生涯奠定了坚实的基础。

### **参考文献**

说明：参考文件不少3个

[1] Sebastián Ramírez. FastAPI Documentation [EB/OL]. (2023-10-26) [2023-10-27]. https://fastapi.tiangolo.com/.

[2] Evan You, et al. Vue.js 官方指南 [EB/OL]. (2023-10-27) [2023-10-27]. https://cn.vuejs.org/guide/introduction.html.

[3] Abraham Silberschatz, Henry F. Korth, S. Sudarshan. 数据库系统概念 (原书第7版) [M]. 北京: 机械工业出版社, 2021.

[4] Python Software Foundation. Python 3.11.6 documentation - asyncio [EB/OL]. (2023-10-27) [2023-10-27]. https://docs.python.org/3/library/asyncio.html.

[5] W3C. Server-Sent Events - W3C Recommendation [EB/OL]. (2015-02-03) [2023-10-27]. https://www.w3.org/TR/eventsource/.