<白云社区>

软件架构文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 13/12/2022 | 1.0 | 技术原型迭代完成后的产品 | 吴基洋，朱楷文，陈梓钊，龚子钦，程凯文 |
| 13/1/2023 | 2.0 | 构造迭代完成后的产品 | 吴基洋，朱楷文，陈梓钊，龚子钦，程凯文 |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 4

3.1 概述 4

3.2 在构架方面具有重要意义的设计包 4

4. 进程视图 4

5. 部署视图 4

6. 实现视图 5

7. 技术视图 5

8. 数据视图（可选） 5

9. 核心算法设计（可选） 5

10. 质量属性的设计 5

软件架构文档

# 简介

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

## 参考资料

无。

# 用例视图

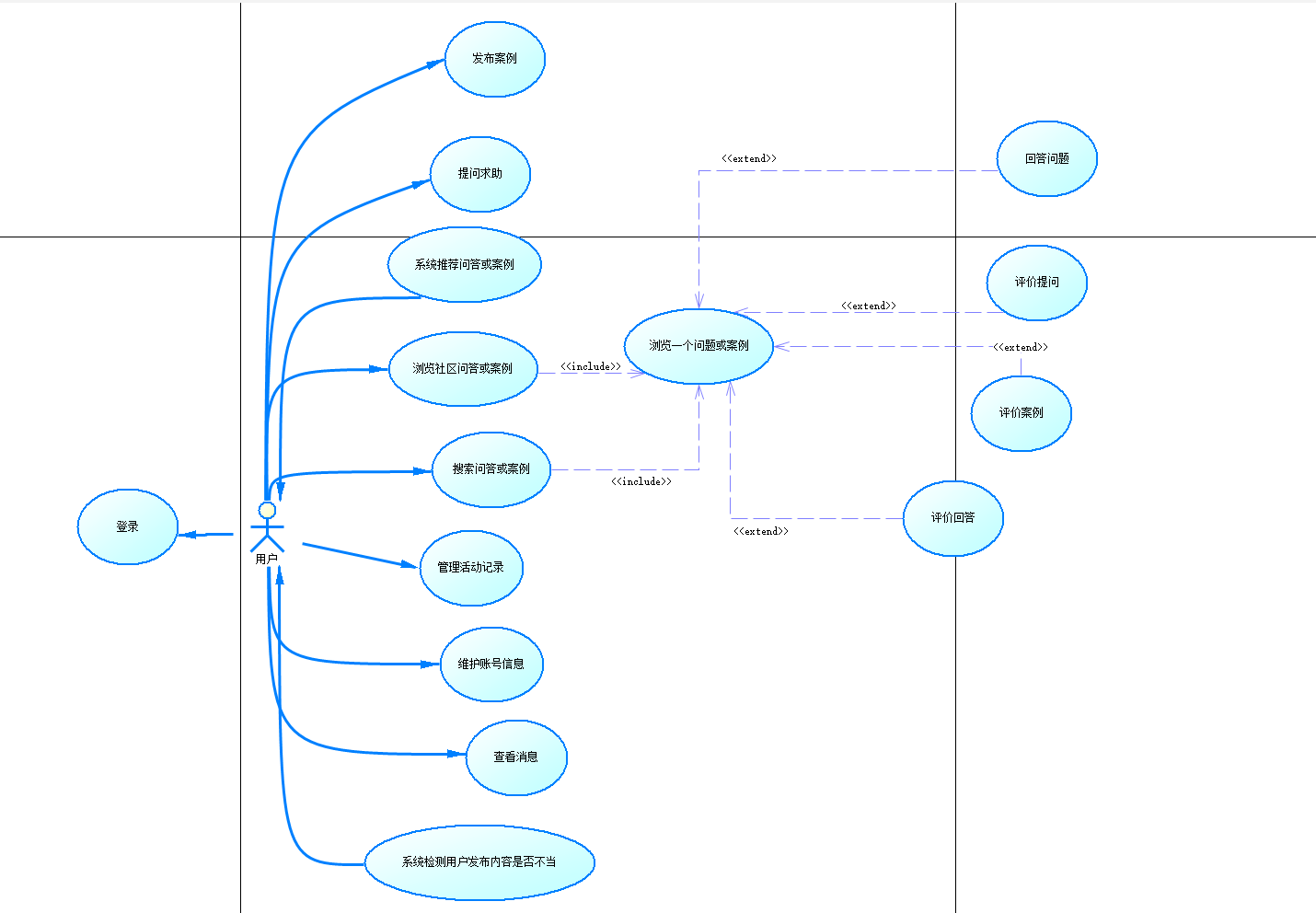
**

图 1 用例视图

用例视图如图1所示，解释如下：

1. 登录：注册一个账号后登录，这是使用社区的前提。
2. 发布共享信息：用户可以共享自己所患疾病的各种信息作为案例，内容包括但不限于病史、诊断、医疗方案、治疗方法、医院和医生、预防措施等信息。
3. 提问求助：用户发起提问，在社区中进行求助。
4. 系统推荐问答或案例：系统根据用户的偏好向用户推荐其可能感兴趣的问答或案例。
5. 浏览社区问答或案例：用户浏览社区根据算法推荐的问答和案例。
6. 搜索问答或案例：用户搜索与某文本相关的问题或回答。
7. 管理活动记录：用户查看并管理自己的提问和回答记录，以及对问答或案例的评论。具体包括：修改提问（仅限问题提出24小时内），关闭提问（指不允许他人再进行回答），删除提问，删除对问题的回答，修改回答，删除回答，删除评论。
8. 维护账号信息：编辑和修改个人信息，如用户名、头像、个性签名、病史等；管理个人的关注列表，关注的问答和收藏的案例。
9. 查看消息：查看系统通知、被赞、被评论、被回答、被私信等各种用户收到的消息。
10. 系统检测用户发布内容是否不当：系统自动检测用户发布的案例、提问、回答或评论是否不当，如广告、灌水、违法等内容。
11. 浏览一个问题或案例：用户浏览一个问题及其所有的回答（如果存在）或一个案例。
12. 回答问题：该用例扩展了“浏览一个问题及其回答”用例。用户对提问进行回答。
13. 评价提问：该用例扩展了“浏览一个问题或案例”用例。用户对某个问题进行点赞或收藏。
14. 评价案例：该用例在“评价问答或案例”扩展点扩展了“浏览一个问题或案例”用例。用户对某个案例进行点赞或收藏。
15. 评价回答：该用例在“评价问答或案例”扩展点扩展了“浏览一个问题及其回答”用例。用户对某个回答进行点赞、收藏、点踩等。

# 逻辑视图

## 概述

本应用的逻辑架构采用3 Tiers风格，分为表示层、业务逻辑层、数据访问层三层，如图2所示，解释如下：

1. 表示层

表示层负责显示用户界面，与用户进行交互，将用户请求发送给业务逻辑层，处理并展示业务逻辑层返回的结果，如图3所示。

1. 业务逻辑层

业务逻辑层负责处理用户请求，执行业务逻辑，如图4所示。前端服务器与表示层交互，处理表示层发送的请求，并调用后端服务器对应的Restful API接口进行处理。后端服务器与数据访问层交互，执行各项业务逻辑。

1. 数据访问层

数据访问层负责处理数据的存储与读取，包括访问核心数据库和访问搜索引擎，如图5所示。

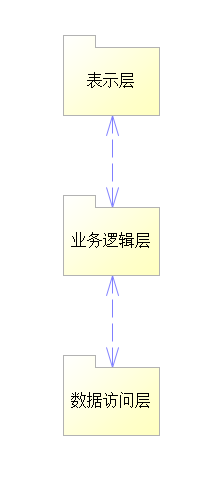


图 2 逻辑视图概览



图 3 表示层

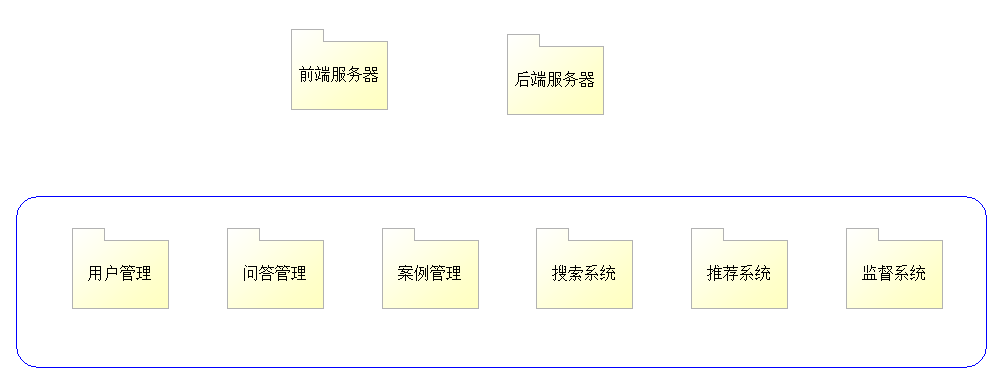


图 4 业务逻辑层

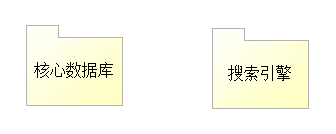


图 5 数据访问层

## 在构架方面具有重要意义的设计包

* 后端服务器控制包

该包用于提供与前端与后端服务器进行交互的 Restful API 接口。该包接收前端发送的请求，调用业务逻辑层中对应的包来执行业务逻辑，并将执行结果返回给前端。

重要的类：

* + 用户控制类：提供与用户管理有关的 Restful API 接口，处理与用户管理有关的请求。
  + 问题控制类：提供与问题管理有关的 Restful API 接口，处理与问题管理有关的请求。
  + 案例控制类：提供与案例管理有关的 Restful API 接口，处理与案例管理有关的请求。
* 数据库访问包

该包用于进行数据库访问，处理数据的读取与存储。

重要的类：

* 用户数据访问类：访问用户数据库中的数据。
* 问题数据访问类：访问问题数据库中的数据。
* 案例数据访问类：访问案例数据库中的数据。

# 进程视图

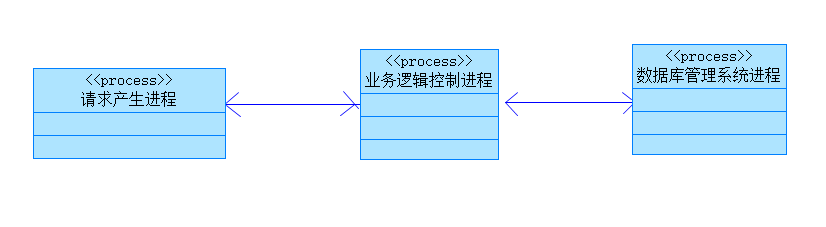


图 6 进程视图

系统主要分为三个进程：请求产生进程，业务逻辑控制进程和数据库管理系统进程。各进程的任务及其之间的信息交互如图6所示，解释如下：

* 请求产生进程：运行在用户访问应用时使用的客户端浏览器，产生查看内容、编辑内容、发布内容、删除内容等请求。
* 业务逻辑控制进程：接收来自请求产生进程的 HTTP 请求，执行业务逻辑的业务处理任务和数据库持久性操作的数据持久化任务，向数据库管理系统进程发送任务并取回结果，然后将业务处理结果组装成 HTTP 响应发送回浏览器。
* 数据库管理系统进程：运行在数据库服务器上，接收来自业务逻辑控制进程的任务，对储存在数据库中的数据进行增删改查操作，并向业务逻辑控制进程返回结果。

# 部署视图

体系结构的物理架构是描述典型平台配置的各种物理节点以及从流程视图到物理节点的任务分配，本应用采用B/S架构风格进行部署，如图7所示，解释如下。

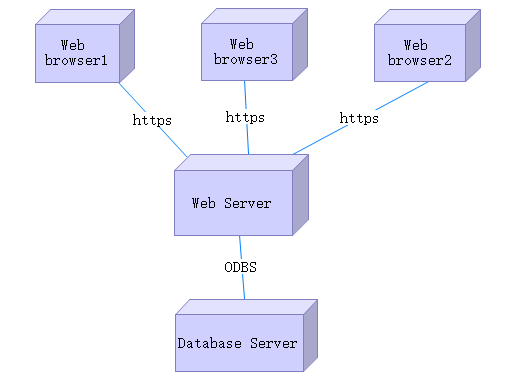


图 7 部署视图

1. Web Browser

用户通过Web浏览器访问网站应用，浏览器（客户端）通过互联网使用https协议向Web服务器发送任务请求并将得到的结果进行显示。

1. Web Server

Web服务器负责接受和处理Web浏览器发送的任务请求并分配任务给数据库服务器处理，接收数据库服务器对任务的处理结果并将响应结果返回给Web浏览器。

1. Database Server

数据库服务器是用来储存用户各种信息，它通过ODBC协议来和Web服务器交互信息，接收来自Web服务器的任务处理请求，执行该任务并将执行结果返回给Web服务器。

# 实现视图

我们基于web的病友互助社区在客户端不需要安装包，仅服务器端有安装包。

在服务器端，软件分为前端，后端，数据库三部分，这三部分的安装包及其之间的依赖关系如图8所示，解释如下：

* 前端：有 Vue, Element-Plus, Axios 三个安装包，其中Element-Plus 和 Axios安装包依赖Vue安装包；
* 后端：有 Tomacat, Apache Shiro, SpringData, SpringBoot四个安装包，其中Tomacat, Apache Shiro 和 SpringData安装包依赖 SpringBoot安装包；
* 数据库：有Mysql安装包。

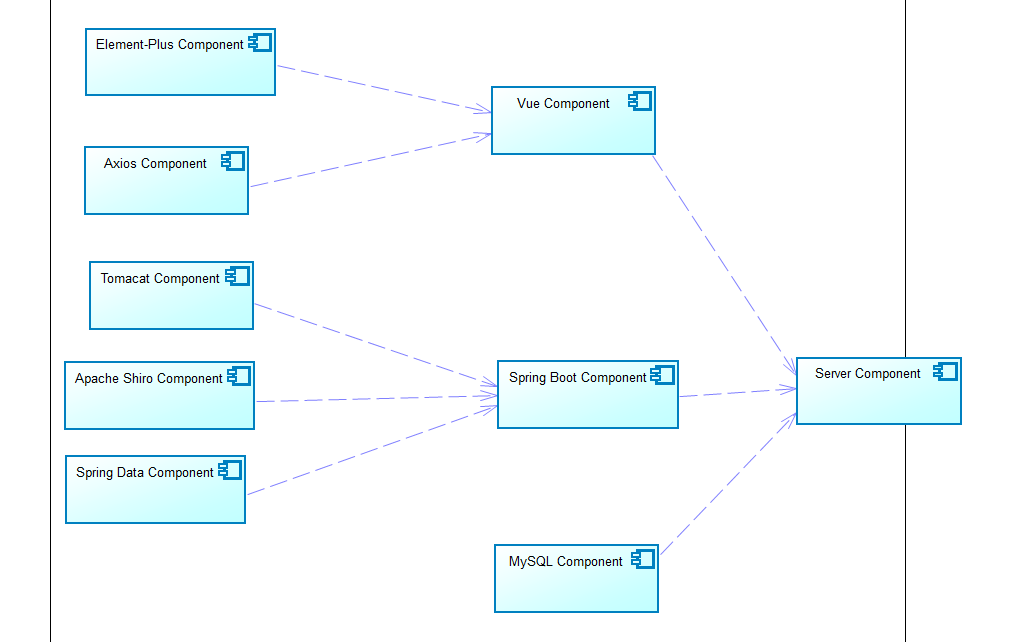


图 8 实现视图

# 技术视图

技术视图如图9所示，解释如下：

* 编程语言：JavaScript, Java.
* 开发工具：IntelliJ IDEA, Visual Studio Code.
* 框架：
* 前端使用 Vue, Element-Plus；
* 后端使用 Spring Boot，Tomacat，Apache Shiro，SpringData.
* 数据库：MYSQL.

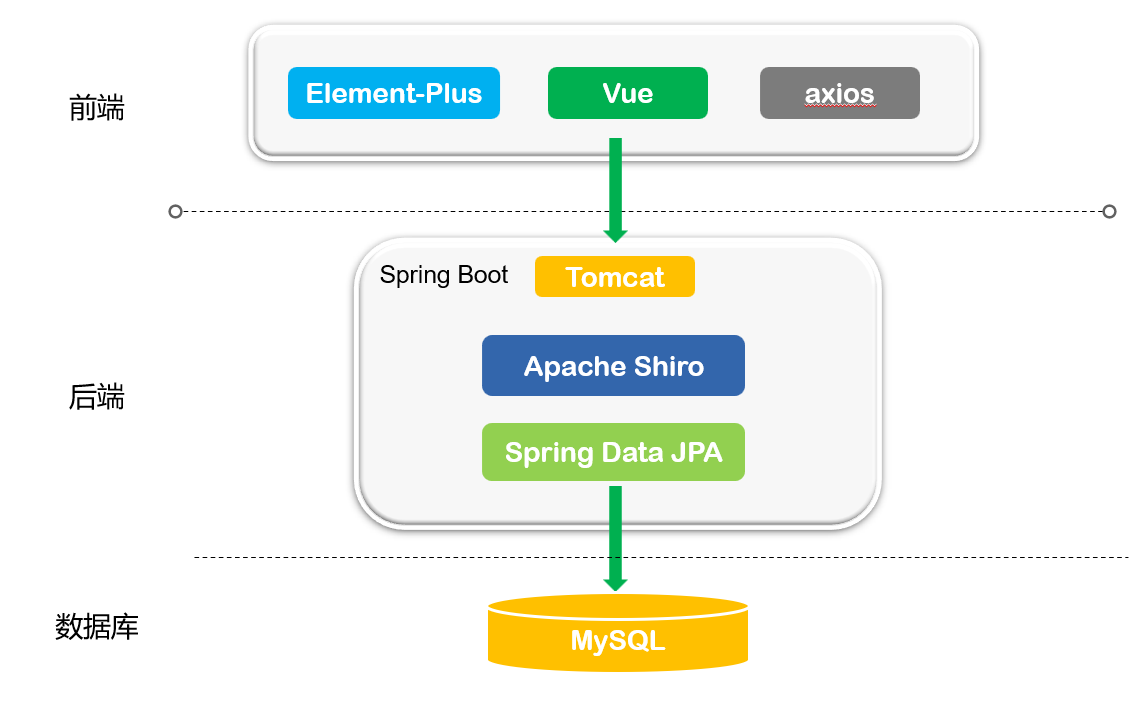


图 9 技术视图

# 数据视图（可选）

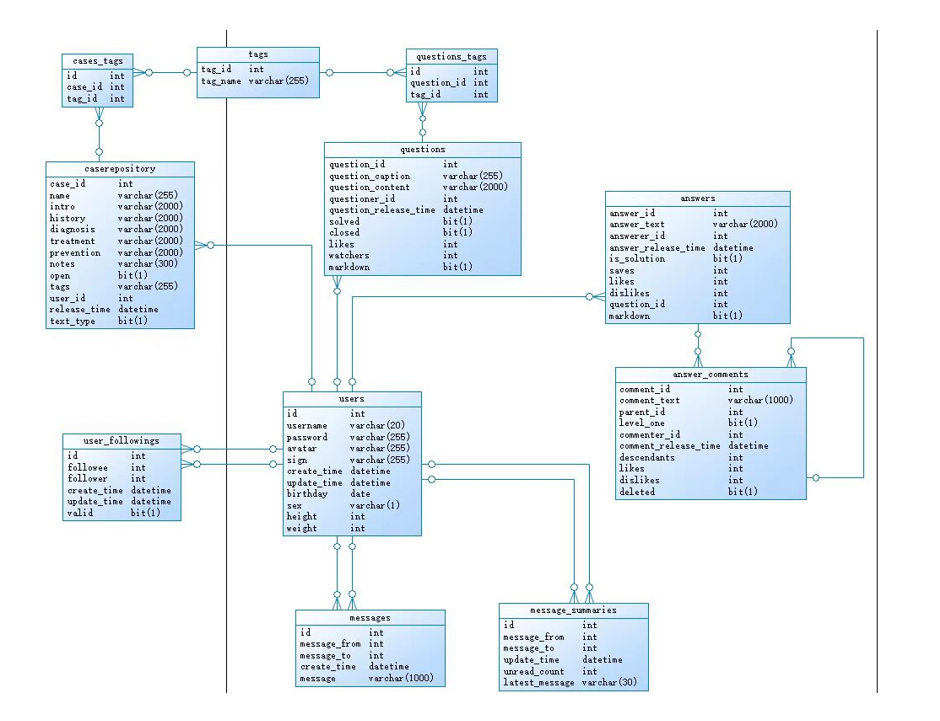


图 10 数据视图

本应用的数据库目前包括十一张表，表之间通过外键相互联系，如图10所示，解释如下：

* answer\_comments：储存所有回答的所有评论
* answers：储存所有回答，通过question\_id与问题相联系。
* caserepository：储存所有案例。
* cases\_tags：储存案例与标签之间的映射。
* questions：储存所有问题。
* questions\_tags：储存问题与标签之间的映射。
* tags：储存所有标签。
* user\_followings：储存用户关注用户的信息。
* user\_message\_summaries：储存用户之间最新的私信消息。
* user\_messages：储存用户之间发送的信息。
* users：储存用户所有资料。

# 核心算法设计（可选）

本应用的核心算法包括内容检测算法和推荐算法，暂定大体框架如下。

内容检测算法：分为图片检测与文本检测。图片检测采取多层卷积网络结构，最后输出该图片是否违规的概率值；文本检测采取单层卷积网络，对语料进行integer vectorization处理后进行计算，最终同样输出违规的概率。

推荐系统算法：采取双层rl智能体的结构，首先由low-level智能体根据环境给出的状态推荐一个标签，然后由high-level智能体从该标签下的案例、问答或用户中推荐一项内容给用户。

# 质量属性的设计

* 性能：查询数据库时利用主键进行加速，还通过外键和中间表连接各个数据库，提高了查询速度；使用了成熟的Spring Boot框架，确保了算法的正确快速。
* 可扩展性：前端使用 Vue 框架的单文件组件，方便添加更多的组件而不影响已有的结构；后端使用Spring Boot框架，各个模块的业务逻辑分开，并进一步分为 Controller 层、Service 层和数据访问层，从而可以方便地添加其他功能而不影响其他部分。
* 可靠性：前后端相互分离，可以一定程度上保证独立性，能够有效避免二者之间错误的传递导致整个系统崩溃；前端设计时充分考虑了可能的错误情况（包括用户的错误操作和服务器的错误）并进行了异常处理；后端使用了成熟的Spring Boot框架，确保可靠性。
* 易用性：将用户定位于外行型，界面设计简洁易懂，对用户知识没有太多的要求，并向市场上流行软件的界面看齐从而更容易符合大多数人的习惯。
* 可移植性：前端界面设计时充分考虑了浏览器版本问题，例如设置的 CSS 样式属性都是各浏览器各版本普遍支持的；后端使用了成熟流行的 Spring Boot 框架，确保可移植性。