<LifeLogger>

软件架构文档

版本 <1.5>

[注：用方括号括起来并以蓝色斜体（样式=InfoBlue）显示的文本，它们用于向作者提供指导，在发布此文档之前应该将其删除。按此样式输入的段落将被自动设置为普通样式（样式=Body Text）。]

[要定制 Microsoft Word 中的自动字段（选中时显示灰色背景），请选择 File>Properties，然后将 Title、Subject 和 Company 等字段替换为此文档的相应信息。关闭该对话框后，通过选择 Edit>Select All（或 Ctrl-A）并按 F9，或只是在字段上单击并按 F9，可以在整个文档中更新自动字段。对于页眉和页脚，这一操作必须单独进行。按 Alt-F9，将在显示字段名称和字段内容之间切换。有关字段处理的详细信息，请参见 Word 帮助。]

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <16/4/2023> | <1,0> | <初始文档> | <杨征，闫景升，张馨匀，刘洋> |
| <12/5/2023> | <1,5> | <后端技术框架改变> | <杨征，闫景升，张馨匀，刘洋> |
|  |  |  |  |
|  |  |  |  |

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

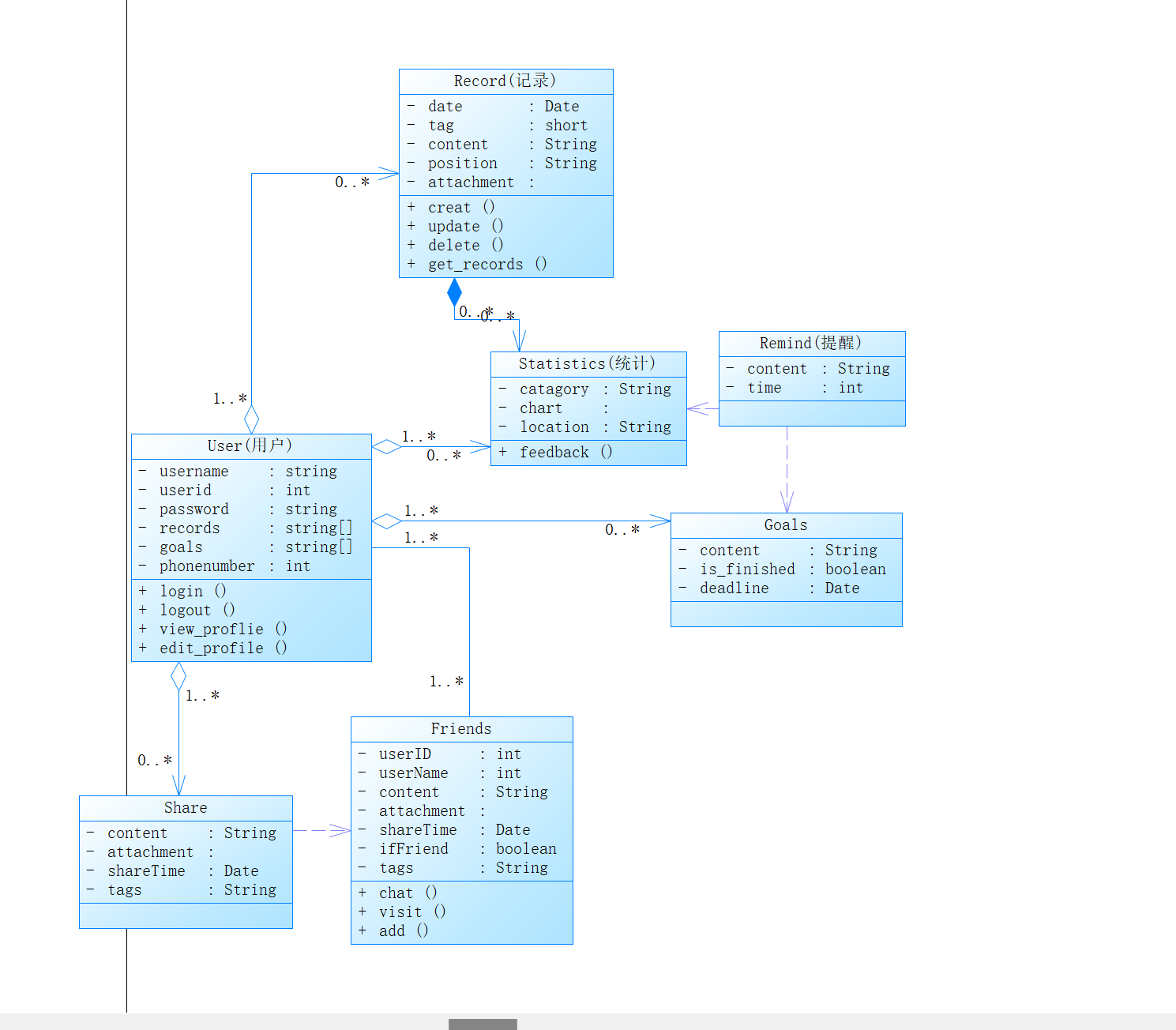
软件架构文档

# 简介

## 目的

本文档将从构架方面对系统进行Life Logger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。

# 用例视图

**

# 逻辑视图

逻辑架构图包括以下几个部分：

1. 用户界面层：负责展示APP的各项功能，包括日记记录、生活规划、事件计时、事件统计和好友圈分享等。
2. 业务逻辑层：负责处理用户的请求，如添加、修改、删除日记，创建生活规划，开始/停止事件计时，统计事件数据，发布/查看好友圈动态等。
3. 数据访问层：负责与数据库进行交互，存储和读取用户数据。
4. 数据库层：存储用户的日记、生活规划、事件计时、事件统计和好友圈分享等数据。

## 概述

整体采用分层开发的风格，将组件界面和服务分开开发。

## 在构架方面具有重要意义的设计包

主要采用Typescript和Javascript两种语言，重要的设计包包括:

用户界面包：

components：

各个功能对应的组件类:日记界面类，记录计时类，登录框类，朋友圈卡片类等

screens：

各个功能对应的界面类，如日记界面类、生活规划界面类、事件计时界面类等

navigation：配置侧边栏、下边栏等，实现界面跳转

业务逻辑包：

data：存放数据接口

l18n:支持多语言转化

service：存放所有的服务

主要采用Python编写，重要的设计包包括:

业务逻辑包：

请求处理类views：

对各个界面发送网络请求的处理

配置类urls&apps：

各个功能对应的请求的url地址

配置数据库

数据库包:

views:负责与数据库进行交互，存储和读取用户数据

models：可视化展示数据库数据

测试包：

test：存放对功能的测试

# 进程视图

1. 消息传递：进程之间通过发送和接收消息来进行通信。例如，用户界面进程可以发送一个消息给业务逻辑进程，请求添加一条日记；业务逻辑进程处理完请求后，再发送一个消息给用户界面进程，通知其操作结果。
2. 中断：当某个事件发生时，可以触发一个中断，通知相关进程进行处理。例如，当用户点击“保存”按钮时，可以触发一个中断，通知业务逻辑进程保存当前的日记。
3. 会合：多个进程在某个时间点或条件下同时到达一个预定位置，进行数据交换或协调操作。例如，用户界面进程和业务逻辑进程可以在某个时间点会合，交换当前的日记数据。

# 部署视图

在单个服务器配置中，APP的所有组件都运行在同一个物理节点上。这个节点可以是一台计算机，具有足够的CPU和内存资源来支持APP的运行。这台计算机可以通过网络连接到互联网，以便用户能够访问APP。

在多服务器配置中，APP的组件可以分布在多个物理节点上。这些节点可以是多台计算机，它们通过网络连接在一起，形成一个集群。每台计算机都具有足够的CPU和内存资源来支持APP的运行。这些计算机可以通过负载均衡器进行负载均衡，以便在高负载情况下保证APP的可用性。

在进程视图中，各进程可以映射到不同的物理节点上。例如，前端用户界面进程可以映射到用户设备上，后端服务器进程可以映射到服务器集群中的某台计算机上。这样，当用户访问APP时，他们的请求会被发送到后端服务器集群中的某台计算机上进行处理。

# 实现视图

APP的整体结构包括前端用户界面，后端服务器和数据库。前端用户界面负责展示APP的功能，如日记记录，生活规划，事件计时，事件统计以及好友圈分享等。后端服务器负责处理用户请求和数据存储。数据库负责存储用户数据。

在实现模型中， APP将前端用户界面分解为视图层，控制层和模型层。视图层负责展示数据，控制层负责处理用户交互，模型层负责管理数据。后端服务器可以分解为应用层和数据层。应用层负责处理用户请求，数据层负责与数据库交互。

在构架方面具有重要意义的构件可能包括用于处理用户身份验证和授权的安全组件，用于处理数据同步的同步组件，以及用于支持多语言的本地化组件等。

# 技术视图

前端：采用React框架，用Javascript或Typescript两种编程语言；

后端:利用Python采用Django框架

# 数据视图（可选）

# 核心算法设计（可选）

# 质量属性的设计

[说明软件架构如何促成诸如性能、可扩展性、可靠性、易用性、可移植性等所有系统能力（而非功能）的实现。如果这些特征具有特殊的意义（例如在安全性或保密性方面的意义），则应该对它们进行详细的说明。]

前端采用React Native框架，具有跨平台性、高复用性、易扩展性的优势

后端采用Django框架，具有易学习性、高安全性、易扩展性的优势