Pose App

组号: A1

成员:

- 聂礼昂
- 金民周
- 赵培源

时间: 2024 年 12 月 26 日

制作于 LaTeX

目录

1 交付产品 **3**

1.1 项目简介 . . . 3
1.2 项目背景 . . . 3
1.3 使用方法 . . . 3
1.4 访问方式 . . . 3
1.5 管理员账号和密码 . . . 4
1.6 代码仓库 . . . 4

2 产品目标 **4**

2.1 功能目标 . . . 4
2.2 性能要求 . . . 5

3 开发组织管理 **5**

3.1 过程管理 . . . 5
3.2 人员分工 . . . 5
3.3 开发环境 . . . 6
3.4 配置管理 . . . 6

4 系统设计 **6**

4.1 整体概述 . . . 6
4.2 前端交互 . . . 8
4.2.1 登陆注册 . . . 8
4.2.2 主页 . . . 8
4.2.3 检测 . . . 9
4.2.4 待办事项 . . . 9
4.2.5 朋友圈 . . . 9
4.2.6 数据统计页面 . . . 12
4.2.7 设置页面 . . . 13
4.3 后端模块 . . . 13
4.3.1 models 模块 . . . 14
4.3.2 API 接口模块 . . . 14
4.4 数据库模块 . . . 15
4.5 坐姿检测模块 . . . 16
4.6 接口规范 . . . 17

5 重点和难点问题及其解决方法 **18**
5.1 前端 18
5.2 检测 18

6 测试总结 **19**
6.1 功能测试 19
6.1.1 自动化测试 19
6.1.2 桌面端测试 19
6.2 性能测试 20
6.2.1 性能调优 20
6.3 安全测试 21
6.3.1 SQL 注入测试过程 21
6.3.2 防范措施 21
6.3.3 敏感信息处理 21

7 系统部署 **22**
7.1 部署方法 22
7.2 部署流程 22
7.3 环境配置 22

1 交付产品

1.1 项目简介

Pose App 是一款专为桌面端设计的工作应用，旨在帮助用户改善坐姿，提升工作效率。该应用融合了先进的 ML 坐姿检测技术 Media_pipe，提供多种实用功能，助力用户在学习和工作时保持良好的坐姿习惯。

Pose App 旨在通过科学的坐姿监测和实用的工作管理工具，帮助用户养成健康的工作习惯，提升工作效率。让我们一起开始使用 Pose App，关注自己的坐姿，提升工作体验！

1.2 项目背景

随着信息技术的发展和办公自动化水平的提高，越来越多的工作被迁移到电脑前完成。这种长时间的坐姿工作使得办公室工作者面临着各种健康问题，尤其是不良的坐姿导致的脊椎、颈部和眼睛的疲劳与伤害。因此，开发一个能够实时监测并纠正办公坐姿的系统显得尤为重要。Pose App 正是在这样的背景下应运而生，旨在通过科学的坐姿监测和实用的工作管理工具，帮助用户养成健康的工作习惯，提升工作效率。

1.3 使用方法

和本项目针对 Windows 笔记本用户，在使用时，建议眼距离屏幕一臂远，腰部挺直，处于正确坐姿下来使用本 App。

图 1: 正确坐姿注：本应用不强制要求您使用电脑支架

1.4 访问方式

- **官方网站**：本项目可以从官方网站访问`http://poseapp.top/`，下载安装程序进行安装。

- **GitHub 仓库**:本项目可以从仓库的 deploy 分支`https://github.com/LeON-Nie-code/pose_app`，下载安装程序进行安装。
- **桌面端**：打包的可执行文件及安装程序见附件。

1.5 管理员账号和密码

- **管理员账号**：test_user_0
- **密码**：123456

1.6 代码仓库

- **代码仓库**：`https://github.com/LeON-Nie-code/pose_app`

2 产品目标

2.1 功能目标

- **计时功能：**点击“开始”后，应用将开始计时，记录你在学习或工作过程中的时间。
- **坐姿检测：**在使用过程中，Pose App 会实时监测你的坐姿情况。完成工作后，你可以点击“停止”，应用将生成一条记录，详细展示你在这段时间内的坐姿表现，包括：
  1. 正常坐姿的持续时间
  2. 歪头，低头，弓腰等不良坐姿的时间
  3. 视线未在屏幕上的时间
- **统计分析：** Pose App 提供每日坐姿统计功能，帮助你了解自己在一天中的坐姿总体情况，便于及时调整和改善。
- **待办事项：**应用内置待办事项功能，方便你记录和管理工作任务，提高工作效率。
- **社区功能：**用户可以通过应用发送帖子，浏览帖子，与同事或朋友分享工作进展和心得。
- **用户管理：**用户可以进行注册、登录，修改个人信息等。
- **排行榜：** Pose App 提供每日专注时间的排行榜。

2.2 性能要求

系统响应时间低于 3 秒，支持同时在线用户数达到 50。

1. **前端服务：**
   (a) 按钮的响应不超过 1 秒。
   (b) 页面跳转响应时间不超过 2 秒
2. **后端服务：**
   (a) 普通 API 响应不超过 1 秒，涉及图片的响应不超过 3 秒
   (b) 进行数据库操作的平均时间不超过 0.5 秒。
3. **本地检测服务：**
   (a) CPU 占用不影响正常工作。
   (b) 内存占用不超过 300MB。

3 开发组织管理

3.1 过程管理

项目迭代采用 Scrum 模式，时间线如下：

- 10.08–10.17：需求分析与设计
- 10.25–12.18：功能开发和测试
- 12.18–12.24：总体测试，部署与验收

3.2 人员分工

- **前端开发**：
  1. 金民周：登录注册页面，主页，待办提醒，朋友圈，数据统计等
  2. 聂礼昂：检测视频页面，与后端 API 对接等
- **后端开发**：
  1. 聂礼昂：用户管理，帖子，检测记录，待办集，评论模块的实现，及其对应 API 等

- **检测开发**：
    1. 聂礼昂：实现对视频流中姿态的识别与分类。
- **测试**：
    1. 聂礼昂：检测模块的性能测试，后端 API 功能测试，后端 API 性能测试。
- **部署与文档编写**：
    1. 聂礼昂：官网搭建，交付文档。
    2. 赵培源：会议记录，应用打包，docker 部署。

3.3 开发环境

1. 后端开发环境
2. 前端开发环境
    - **操作系统**：Windows 11, Mac OS
    - **技术栈**：Python 3.12, Flask, Flutter，Dart，media_pipe
    - **开发工具**：VS Code, Docker, Postman, ApiFox，Jmeter

3.4 配置管理

使用 Git 进行版本控制，分支策略如下：

- **main 分支**：经测试的稳定版本
- **release 分支**：发布版本
- **feature 分支**：按功能划分的开发版本

4 系统设计

4.1 整体概述

本项目前端使用 Flutter 框架实现 Windows 桌面应用；基于 media_pipe 来在本地实现姿态检测，使用 Flask 代理服务；后端使用 Flask 提供 API 服务，并使用 sqlite 作为服务器。以下是整体架构图。

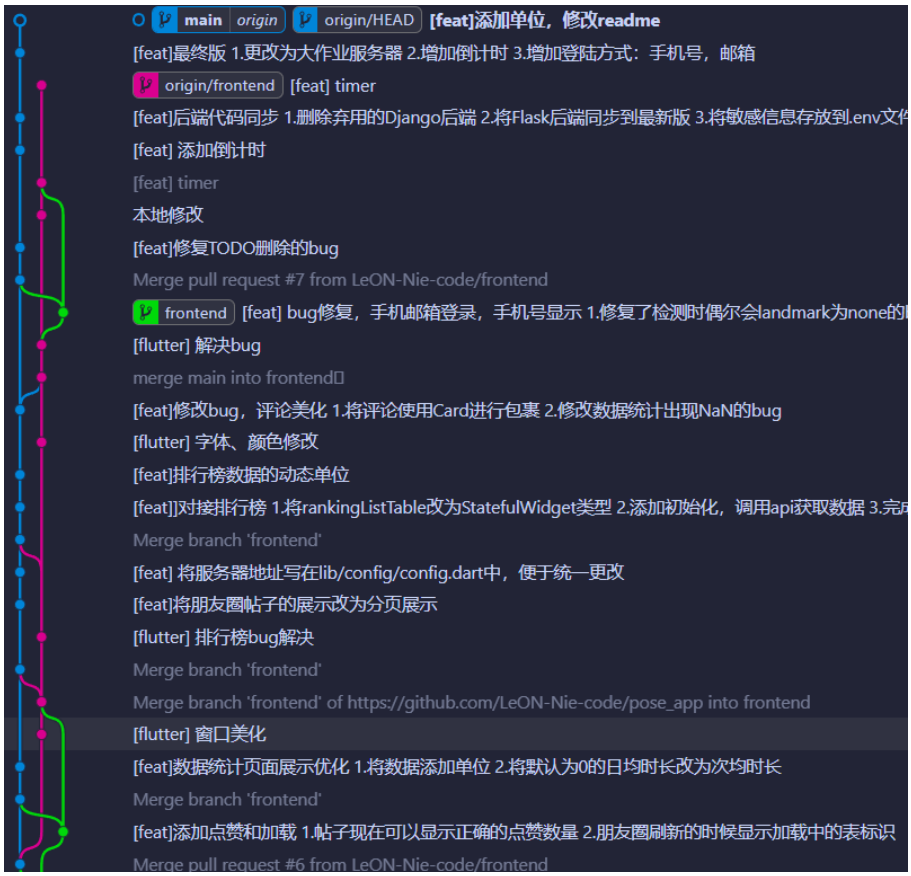
main origin
origin/HEAD
[feat]添加单位，修改readme
[feat]最终版 1.更改为大作业服务器 2.增加倒计时 3.增加登陆方式：手机号，邮箱
origin/frontend
[feat] timer
[feat]后端代码同步 1.删除弃用的Django后端 2.将Flask后端同步到最新版 3.将敏感信息存放到.env文件
[feat] 添加倒计时
[feat] timer
本地修改
[feat]修复TODO删除的bug
Merge pull request #7 from LeON-Nie-code/frontend
frontend
[feat] bug修复，手机邮箱登录，手机号显示 1.修复了检测时偶尔会landmark为none的
[flutter] 解决bug
merge main into frontend
[feat]修改bug，评论美化 1.将评论使用Card进行包裹 2.修改数据统计出现NaN的bug
[flutter] 字体、颜色修改
[feat]排行榜数据的动态单位
[feat]对接排行榜 1.将rankingListTable改为StatefulWidget类型 2.添加初始化，调用api获取数据 3.完成
Merge branch 'frontend'
[feat] 将服务器地址写在lib/config/config.dart中，便于统一更改
[feat]将朋友圈帖子的展示改为分页展示
[flutter] 排行榜bug解决
Merge branch 'frontend'
Merge branch 'frontend' of https://github.com/LeON-Nie-code/pose_app into frontend
[flutter] 窗口美化
[feat]数据统计页面展示优化 1.将数据添加单位 2.将默认为0的日均时长改为次均时长
Merge branch 'frontend'
[feat]添加点赞和加载 1.帖子现在可以显示正确的点赞数量 2.朋友圈刷新的时候显示加载中的表标识
Merge pull request #6 from LeON-Nie-code/frontend

图 2: 分支

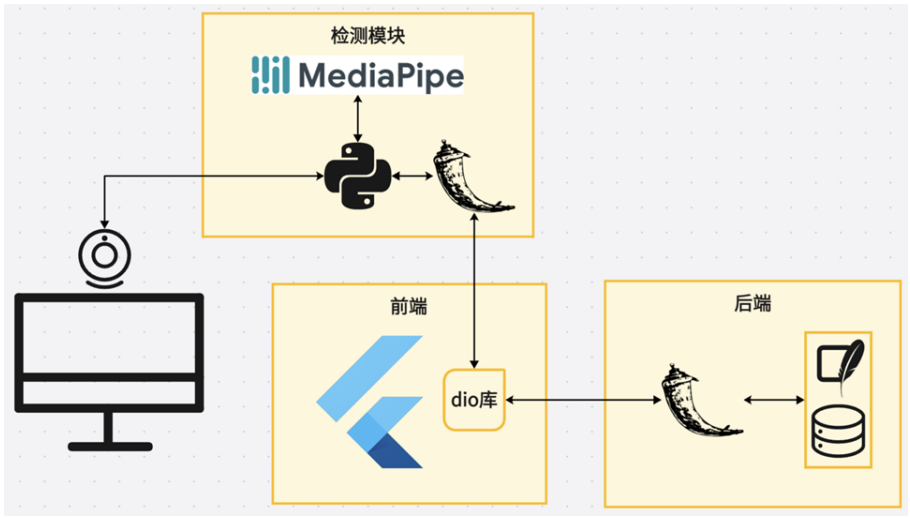
检测模块
MediaPipe
前端
dio库
后端

图 3: 整体架构

4.2 前端交互

界面设计原则：简洁直观，易于操作。

4.2.1 登陆注册

- **登陆**：对输入信息进行验证，调用后端 API，身份验证成功后将会跳转到首页。就可以选择手机登录，邮箱登录。
- **注册**：对用户名，电话号，邮箱信息进行 unique 验证，注册成功后返回登陆页面。

图 4: 登录和注册

4.2.2 主页

- **导航栏**：通过左侧导航栏可以切换不同页面。
- **排行榜卡片**：对今日专注时间最长的 5 名用户进行展示。
- **今日待办卡片**：对日期为今天的待办事项进行展示。
- **数据统计**：对用户过去七天的专注时间的统计。
- **专注栏**：点击开始可以开始专注和坐姿检测，切换“显示”按钮可以选择视频是否显示检测识别细节，以下显示用户的专注记录，滑动滚轮访问更多记录。
- **头像**：点击右上角头像即可显示用户 ID，点击退出账号，即可退出并返回登陆页面。

图 5: 主页

4.2.3 检测

- **切换按钮**：点击“显示”按钮即可切换模式，是否显示检测识别细节。
- **检测窗口**：点击开始弹出检测窗口，实时对视频流进行姿态检测。
- **退出**：点击 close 即可退出并对检测记录进行保存（低于 10 秒不予保存）。
- **更新 ui**：点击黄色的刷新按钮即可获得最新的记录列表。

4.2.4 待办事项

- **日历**：点击不同日期来展示该日期的待办事项
- **TODO 展示**：使用黄色卡片对待办事项进行展示。
- **新增页面**：点击添加跳转到新增 TODO 的页面，输入事项，备注，日期，即可新建 TODO。
- **删除 TODO**：滑动 TODO 卡片即可删除。

4.2.5 朋友圈

- **帖子展示**：自上而下对所有的帖子进行展示。
- **帖子信息**：每个帖子显示用户名内容，日期，图片，评论，点赞数，点击箭头可以显示不同的图片。点击图片可以放大显示。
- **评论**：点击评论按钮弹出 Dialog 来填写评论，点击发送即可完成评论。

图 6: 各种检测结果

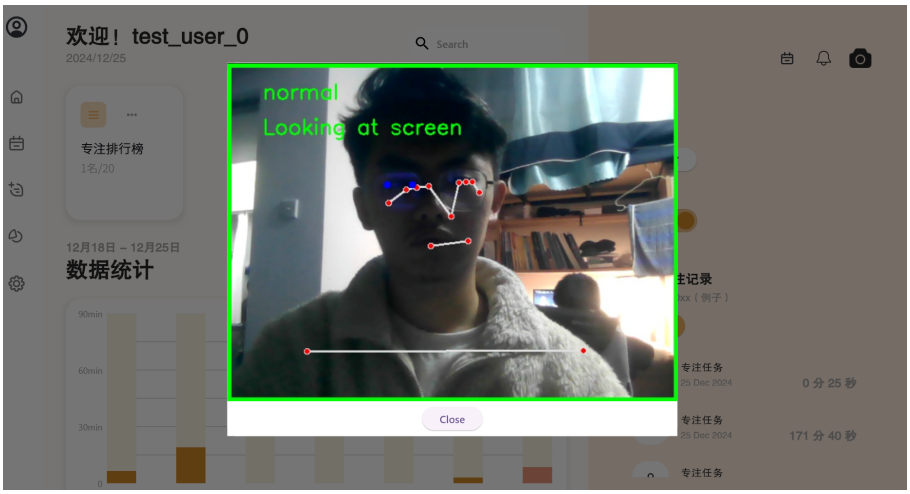

图 7: 坐姿检测

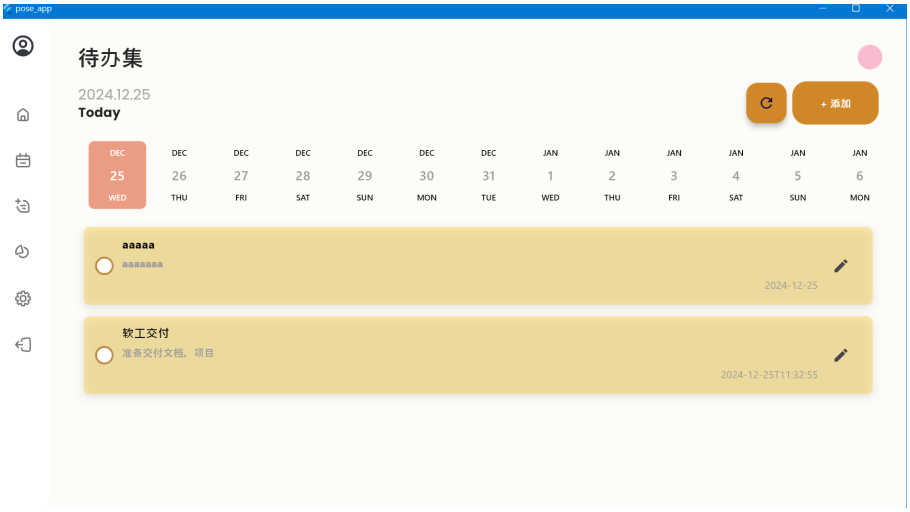

图 8: 待办事项

图 9: 新增待办事项

- **点赞**：点击 like 即可对帖子点赞，同一个用户可以多次点赞同一个帖子。
- **刷新**：点击右上角刷新按钮即可对本页面刷新，获取信息的过程中会显示加载中的标识。
- **发帖**：点击右上角加号按钮会弹出发帖窗口。输入文字，选择图片 (限制最多三张)，点击右上角即可发贴。

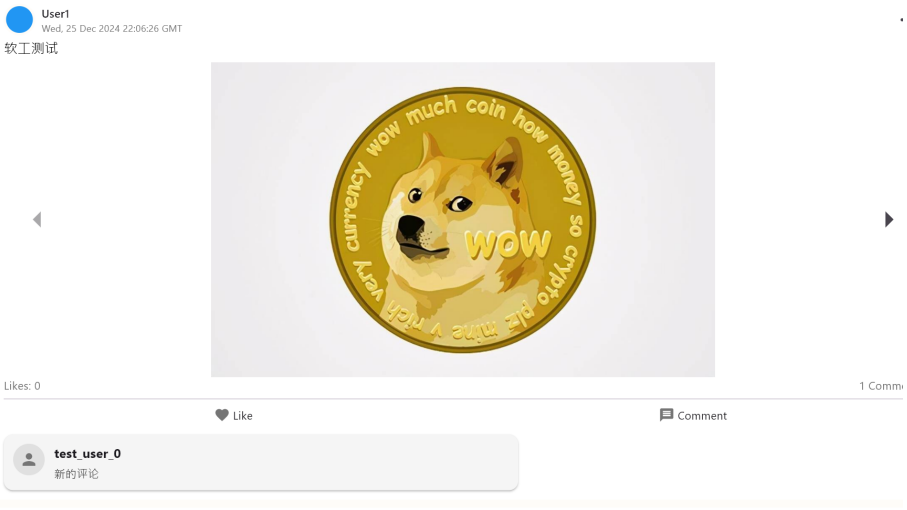

图 10: 朋友圈

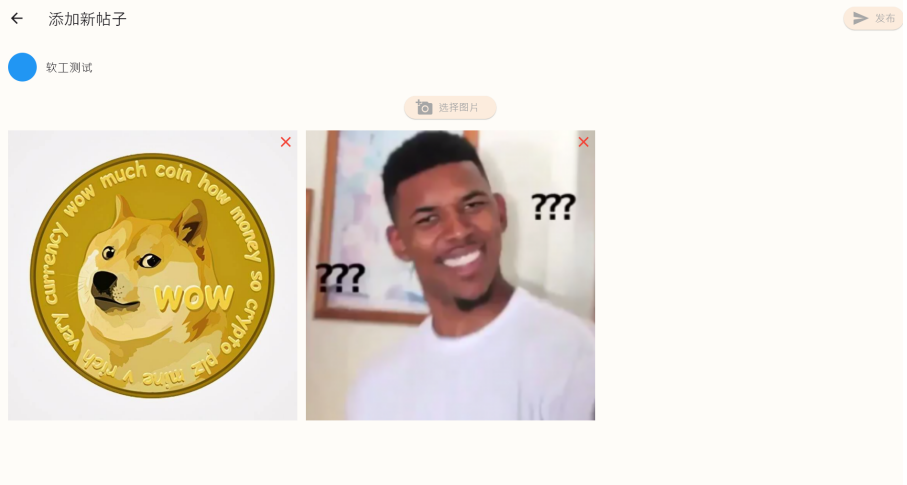

图 11: 新增帖子

4.2.6 数据统计页面

- **累计专注**：对该用户总共的专注次数，专注时长，次均时长进行展示。
- **今日专注**：对该用户今日的专注次数，专注时长，次均时长进行展示。

- **饼状图分析**：对在日历上选择日期当天的数据进行分析展示，使用饼状图显示总时长和异常时长。
- **日历**：今日显示未黄色背景，有专注记录的日期下标有红色小圆圈，选中的日期显示为红色背景。点击左右箭头可以切换月份。
- **刷新**：右上角刷新按钮即可对本页面刷新。

图 12: 数据统计

4.2.7 设置页面

- **个人信息**：对该用户的个人信息进行展示，包括账号 ID，姓名，学校/公司，性别，邮箱，年龄。
- **修改信息**：点击右上角编辑，在弹出 Dialog 中即可对个人信息进行编辑，点击保存即可更新。

4.3 后端模块

采用模块化架构，核心模块包括：

- models 模块
- API 接口模块
- 数据库模块

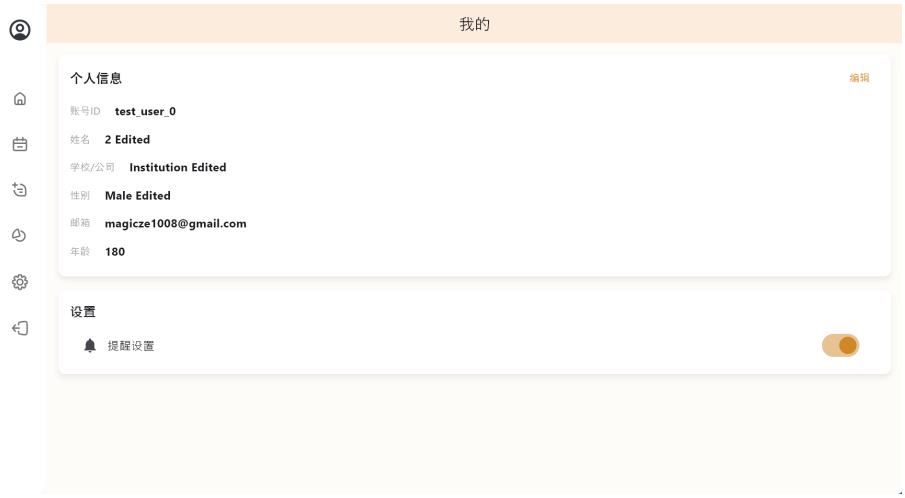

图 13: 个人信息

4.3.1 models 模块

该模块实现了社交类应用的核心功能：

- **用户管理**：User 模型存储用户基本信息。
- **内容发布**：Post 和 Comment 模型支持帖子与评论功能。
- **人体检测记录**：UserDetectionRecord 模型记录用户的人体检测信息。
- **身份验证**：VerificationCode 模型支持手机验证码登录。
- **待办事项**：Todo 模型支持用户的日程管理。

4.3.2 API 接口模块

本后端项目的 API 主要分为以下几个部分

- **用户管理 login/register/user**：用户注册、登录、个人信息查询和修改等，通过 bcrypt 使用一个盐和一个复杂的哈希函数来生成密码的哈希值。通过 JWT 访问令牌来进行身份验证。
- **记录管理 record**：用于检测记录的查询和上传。用户需要提供检测的持续时间、开始和结束时间、体态和眼动次数等数据，且接口仅允许已登录用户访问。
- **帖子管理 post**：用于帖子的发布，获取，评论，点赞，每个帖子最多可上传三张图片。帖子的获取使用分页系统来降低加载时间。
- **待办管理 todo**：用于用户待办事项的添加和获取。
- **好友 friend**：用于好友的添加和获取等，由于前端来不及实现，遂割弃。

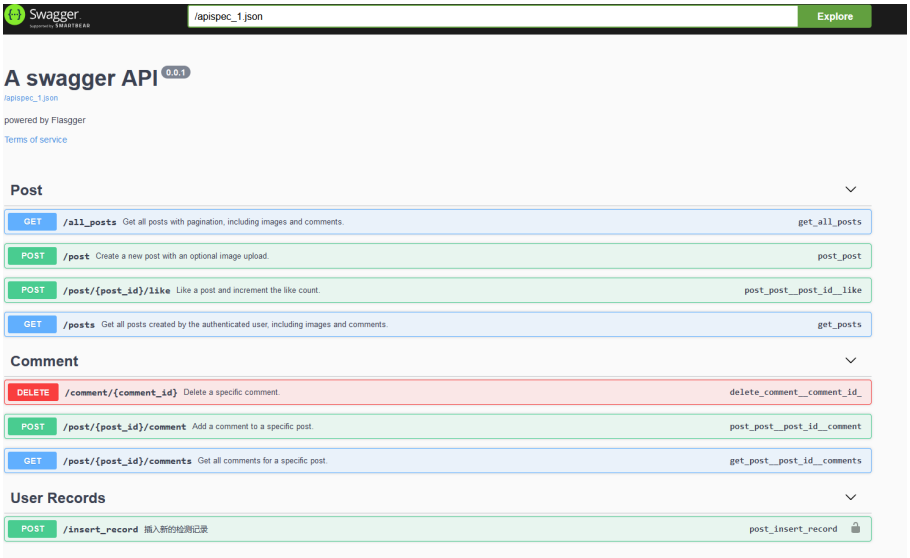

图 14: API

4.4 数据库模块

本系统的数据库模块使用了 SQLAlchemy ORM 与 Flask 框架集成，采用 SQLite 数据库。数据库模型包括多个表，涵盖了用户、文章、评论、待办事项等多个功能模块。以下是主要模型的介绍：

- **User** 表：用于存储用户基本信息，包括用户名、电子邮件、密码、电话等。每个用户可以拥有多篇文章和评论记录，以及待办事项。
- **UserDetectionRecord** 表：记录每次用户检测的数据，包括检测开始时间、结束时间、持续时间，以及用户的姿势和眼睛状态等数据。该表与用户表通过外键关联。
- **VerificationCode** 表：存储用户的短信验证码以及验证码的创建时间，包含一个过期检查的方法。
- **Post** 表：用于存储用户发布的文章，包含标题、内容、发布时间等信息。每篇文章最多可包含三张照片，存储为 BLOB 类型。同时，文章支持点赞和评论。
- **Comment** 表：用于存储用户对文章的评论，包括评论内容、评论时间、关联的文章和用户信息。
- **Todo** 表：用于记录用户的待办事项，包括标题、备注、提醒时间等信息。
- **Friendship** 表：用于存储用户之间的好友关系，包括好友的状态（如待处理、已接受、已屏蔽）。

此外，数据库使用 SQLAlchemy 的关系定义来建立不同表之间的关联，如用户与检测记录、用户与文章、用户与评论等。

模型中的时间字段使用了 SQLAlchemy 的 DateTime 类型，并设置了默认值，便于记录创建时间。所有的时间字段（如创建时间、发布时间、评论时间等）都使用了 UTC 时间，确保系统的时间一致性。

数据库的表关系由外键和关系定义（如 `db.relationship`）确保数据的一致性和完整性，支持方便的查询和数据操作。

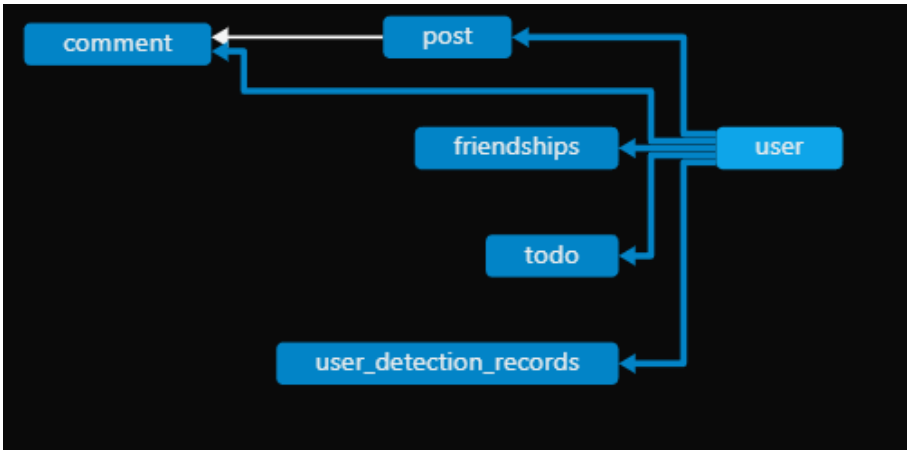

图 15: 数据库

4.5 坐姿检测模块

本模块使用了 MediaPipe 库来进行人体姿势检测，结合 OpenCV 实现视频流的实时处理和姿势分析。坐姿检测模块的核心功能是通过摄像头捕捉用户的实时视频，并利用 MediaPipe 提供的姿势检测模型（Pose）和面部网格模型（Face Mesh）分析用户的坐姿、眼睛位置等信息，进而评估用户的坐姿是否符合健康标准。

模块的工作流程如下：

- **视频流获取**：通过 OpenCV 库从摄像头获取实时视频流，并将其转换为 RGB 格式以供 MediaPipe 处理。
- **姿势检测**：使用 MediaPipe 提供的 Pose 模型对每一帧图像进行处理，识别用户的关键点（例如头部、肩膀、臀部等）。根据这些关键点的位置和角度，判断用户的坐姿是否正常。通过角度计算，将坐姿大致分为：左/右歪头，左/右侧脸，弓腰，低头，抬头，高/低肩，正常等种类。
- **眼睛位置检测**：同时，模块通过 MediaPipe 的 Face Mesh 模型分析用户的面部关键点，眼睛的坐标位置。通过分析面部的朝向，来间接判断视线位置，从而判断用户是否在看向屏幕，帮助判断用户的用眼状态。

- **姿势与眼睛状态记录**：当检测到用户的坐姿发生变化时，模块会记录变化发生的时间，并统计每种坐姿持续的时间。同样，眼睛的状态（是否注视屏幕）也会被记录并持续更新。
- **视频流显示**：处理后的图像通过 OpenCV 在实时视频流中绘制姿势关键点、边框标识和眼睛状态，帮助用户直观地看到当前坐姿的正确性和眼睛位置的健康状态。同时也提供纯净流，不显示检测细节，只通过视频边框颜色来显示坐姿正确与否。
- **性能监控**：模块还提供了一个后台性能监控功能，通过 'psutil' 库实时监控 CPU 使用率、内存占用、网络流量和磁盘读写等信息，确保服务的稳定性。
- **日志模块**：使用'logging' 库来在日志文件中，实时并详细地记录检测过程，便于查看和调试。

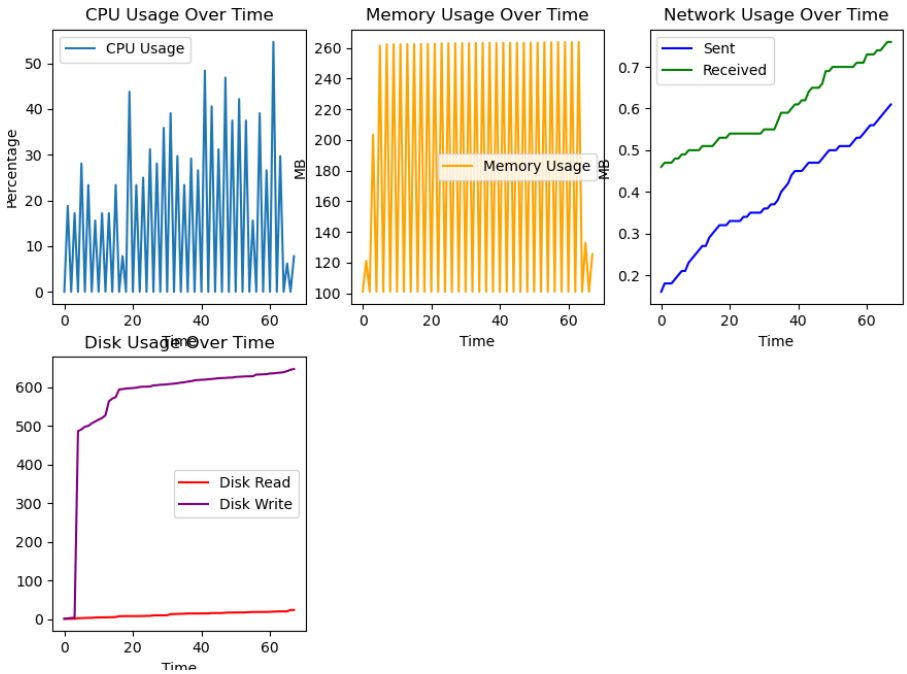

图 16: 检测模块性能测试：基本符合预期要求

4.6 接口规范

本项目使用 swagger api 进行接口的管理和测试。可直接在`http://8.217.68.60/apidocs/`，或`http://118.89.124.30/apidocs/` 对文档进行访问。

本项目在 Postman 以及 ApiFox 均有 Api 目录,`https://api.postman.com/collections/30414775-cc7da313-e3d4-4dc0-b184-d3eccaf717c0?access_key=PMAT-01JFWNRM15SNKPNBTPD8RSK`

5 重点和难点问题及其解决方法

5.1 前端

在前端设计过程中，我们面临了对 Flutter 框架中关键组件的不熟悉问题，这包括但不限于 `Container`、`Scaffold`、`SizedBox`、`Dialog`、`ElevatedButton`、`IconButton`、`Drawer` 和 `SingleChildScrollView`。这种不熟悉导致了在布局和样式调整时的挑战，尤其是在参数配置的一致性上。我们发现，不同组件在尺寸设置、边距（`padding`、`margin`）以及层级关系方面的处理方式存在显著差异，这不仅增加了开发难度，而且容易引发布局超出屏幕边界或不符合设计规范的问题。

为了解决这些问题，我们采取了以下措施：

- 深入学习 Flutter 官方文档，以更好地理解各组件的用途和配置方法。
- 积极参与 Flutter 社区讨论，与其他开发者交流经验和最佳实践。
- 进行组件特定的代码审查，确保布局和样式的一致性。

5.2 检测

在利用 MediaPipe 进行人体姿态识别与坐姿检测的项目中，我们遇到了若干挑战。首先，由于人体姿态的多样性和复杂性，确保算法能够准确识别各种坐姿是一个技术难题。其次，环境光线的变化和背景的复杂性对检测的准确性构成了影响。此外，实时性要求也对算法的效率提出了挑战。

为了解决这些问题，我们采取了以下措施：

- 对算法进行了优化，以提高对不同坐姿的识别能力，包括但不限于正坐、侧坐和斜坐等。
- 计划使用深度学习等方法对坐姿进行分类，但由于样本太小，效果很差。
- 通过并行计算和算法优化，提高了算法的运行效率，确保了实时性的要求。

通过这些努力，我们显著提高了坐姿检测的准确性和可靠性，为后续的应用开发奠定了坚实的基础。

6 测试总结

6.1 功能测试

6.1.1 自动化测试

使用 apifox 和 postman 各个 api 的返回结果和状态码进行断言测试，并进行自动化配置。

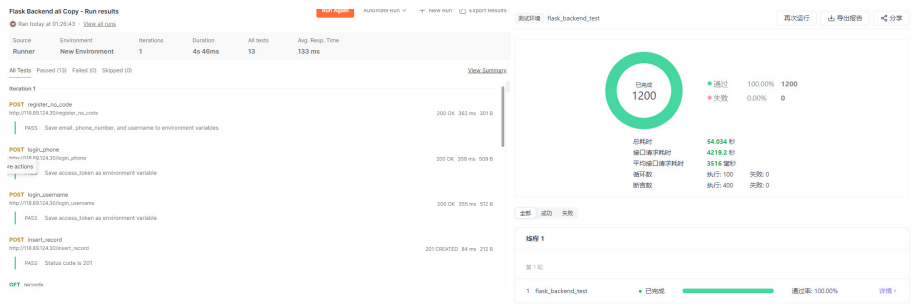

图 17: 自动化功能测试

6.1.2 桌面端测试

对以下功能在不同的 WIndows 设备，不同环境下进行了场景测试

测试用例	**预期结果**	**测试结果**
用户登录	成功跳转到主页	通过
用户注册	成功并返回登录	通过
点击开始专注	返回检测结果的视频	通过
点击专注记录	弹出详细信息窗口	通过
点击 TODO 页面	显示 TODO	通过
点击添加 TODO	添加 TODO 并返回	通过
点击不同日期	显示所选日期的 TODO	通过
滑动删除 TODO	成功删除并更新 UI	通过
点击朋友圈页面	显示帖子	通过
滑动查看	成功显示	通过
点击下一页	显示下一页帖子	通过
点击添加帖子	弹出添加页面	通过
选择图片	正常显示图片	通过
点击发布	成功发布并返回	通过
点击点赞	点赞数加 1	通过
点击评论	弹出评论页面	通过

发送评论	成功发布并显示	通过
点击 TODO 页面	显示 TODO	通过
点击数据统计页面	显示可视化数据	通过
点击个人信息页面	显示个人信息	通过
点击登出账号	退出并返回登录	通过

6.2 性能测试

测试使用 ApiFox 并行测试，测试结果如下

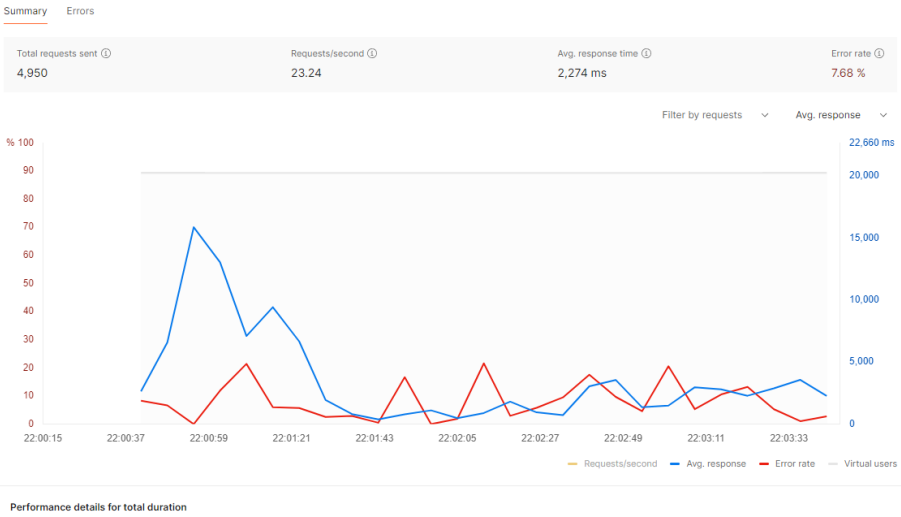

图 18: 性能测试

表 2: 并发量与性能指标

并发量	平均返回时间/ms	50% 返回时间/ms	90% 返回时间/ms	95% 返回时间/ms	错误率
20	1194	1197	1564	1925	2.65%
50	1465	1489	1956	4141	4.78%
100	2,274	2,347	3210	4523	7.68%

6.2.1 性能调优

首次性能测试发现，每次获取 posts 的时间很长，甚至能达到 10 秒左右。经分析发现，这是由于每次获取 posts 都是将数据库中的所有帖子都查询出来并返回，再加上帖子中包含图片信息，因此当数量多起来之后，每次获取的时间就很长。

因此对获取帖子的 API 进行优化，对其进行分页处理，每次只获取一页固定数量的帖子，之后再获取下一页的帖子。同时前端也进行调整，分页展示帖子。通过这种办法，有效降低了获取 post 的时间。

6.3 安全测试

在本系统的后端安全性测试中，特别关注了 SQL 注入（SQL Injection）攻击的防范。SQL 注入攻击是通过将恶意 SQL 代码插入到用户输入字段中，诱使应用程序执行恶意查询的攻击方式。为了确保系统的数据库操作不受到 SQL 注入攻击的影响，进行了以下测试。以下测试语句参考了`https://www.w3schools.com/sql/sql_injection.asp`

6.3.1 SQL 注入测试过程

1. **登录接口测试**：对登录接口进行 SQL 注入攻击，模拟输入：

```
' OR '1'='1'; --
```

该注入字符串尝试绕过身份验证过程，获取未授权访问。经测试返回：”message”: ”Invalid credentials”

2. **用户注册接口测试**：对注册接口输入恶意字符串：

```
' OR '1'='1'; --
```

测试系统是否会创建一个未授权的用户账户。经测试返回：”message”: ”Invalid credentials”

3. **查询接口测试**：通过在查询接口注入恶意 SQL 代码，验证是否会返回不正当的数据，或修改数据库中的数据。

```
' UNION SELECT NULL, username, password FROM users; --
```

尝试从数据库中提取敏感数据，用户名和密码。经测试返回：”message”: ”Invalid credentials”

6.3.2 防范措施

我们在搭建后端的时候，确保了所有数据库操作都通过 ORM（如 SQLAlchemy）进行，使用 ORM 的查询接口来避免直接拼接 SQL 语句，防止 SQL 注入的风险。此外，用户的输入都经过了严格验证和转义，避免恶意代码的注入。

6.3.3 敏感信息处理

本项目使用了 TruffleHog 来扫描代码，发现存有本预计实现短信验证的的 ALIBABA_CLOUD_ACCESS_KEY_SECRET 和 ALIBABA_CLOUD_ACCESS_KEY_ID（此前强制推送到了 GitHub，后续对其销毁），最后将其统一存储在了.env 文件中，并写入.gitignore，避免泄露。。

7 系统部署

7.1 部署方法

1. **前端 + 检测**：使用工具各自打包成一个 exe 文件，接着统一打包成一个自动安装程序。

2. **后端 + 数据库**：各自使用 docker 进行部署，并采用 Docker-Compose 编排所有容器。

7.2 部署流程

1. **detect**：在 `detect` 文件夹下执行以下命令，创建虚拟环境并安装所需依赖：

   ```
   python -m venv venv
   venv\Scripts\activate
   pip install -r requirements.txt
   python app.py
   ```

   这样即可在 `http://127.0.0.1:5000` 提供 API 服务。并且可以在`http://127.0.0.1:5000/apidocs/#/`访问 Swagger 文档。

2. **flutter**：在 `flutter` 文件夹下执行以下命令，启动 Flutter 前端应用：

   ```
   flutter run -d windows
   ```

   此命令将启动 Windows 平台上的 Flutter 前端应用。

3. **my_flask_app**：在 `my_flask_app` 目录下执行以下命令，使用 Docker 启动后端 API 服务：

   ```
   docker-compose build
   docker-compose up
   ```

   这将启动并提供后端 API 服务。Swagger 文档位于`http://118.89.124.30/apidocs/`

7.3 环境配置

- Docker 版本：20.10
- Python 版本：3.12.0
- 数据库：sqlite 8.1