

浙江理工大学

智能软件综合研究 实训报告



学期	2023-2024-1	
专业	计算机科学与技术	
成员	陈昊天	2021329600006
	戴建雨	2021333503068
	周佳羽	2021329621007
	严贺阳阳	2021329621006
	石查分	2020333504025
教师	马明泽、吴婷婷	

目录

1 引言	1
1.1 项目背景	1
1.2 项目目的	1
1.3 项目范围	1
2 项目成员职务及任务分解	1
2.1 成员角色概述	1
2.2 成员职务及任务分解表	2
3 项目的开发计划	2
3.1 开发计划概述	2
3.2 项目开发计划表	3
4 可行性及需求分析	5
4.1 可行性分析	5
4.1.1 技术可行性	5
4.1.2 经济可行性	5
4.1.3 市场可行性	6
4.1.4 可行性风险和挑战	6
4.2 需求分析	7
4.2.1 功能需求	7
4.2.2 数据需求	8
4.2.3 性能需求	8
5 系统设计	9
5.1 系统设计概述	9
5.1.1 软件流程图	9
5.1.2 系统架构图	10
5.2 前端设计	10
5.3 后端设计	11
5.4 数据库设计	13
5.4.1 数据库设计概述	13
5.4.2 数据库 ER 图	13
5.5 安全设计	14
6 模块设计	14
6.1 模块汇总	14
6.1.1 模块汇总表	14
6.1.2 模块框架图	16

6.2 用户子系统的模块设计	16
6.2.1 登录模块	16
6.2.2 注册模块	18
6.2.3 个人中心模块	20
6.3 车辆监控子系统的模块设计	22
6.3.1 车辆列表模块	22
6.3.2 车辆地图模块	24
6.4 疲劳检测子系统的模块设计	26
6.4.1 实时视频监控模块	26
6.4.2 疲劳检测模块	27
6.4.3 警告提示模块	29
6.5 数据统计子系统的模块设计	30
6.5.1 疲劳记录模块	30
6.5.2 统计分析模块	31
6.6 系统管理子系统的模块设计	33
6.6.1 用户管理模块	33
6.6.2 车辆管理模块	34
6.7 数据通信子系统的模块设计	36
6.7.1 全双工通信	36
6.7.2 负载均衡器	36
6.7.3 国际网络优化	37
7 测试设计及测试报告	38
7.1 测试设计概述	38
7.2 测试环境设置	38
7.3 测试用例设计	39
7.3.1 功能测试	39
7.3.2 性能测试	39
7.3.3 安全测试	40
7.3.4 用户体验测试	40
7.4 测试结果与报告	40
7.4.1 测试结果	40
7.4.2 测试报告	41
8 项目日常沟通机制及记录	42
8.1 日常沟通机制	42
8.2 沟通内容记录	42

9 项目管理机制及执行记录	45
9.1 项目管理机制	45
9.2 项目执行记录	45
10 团队成员评价及理由	47
10.1 团队成员评价	47
10.2 成员评价理由	47
参考文献	48

1 引言

1.1 项目背景

随着计算机科学和技术的迅速发展，智能软件在各领域的应用日益广泛。其中，利用计算机视觉技术进行疲劳驾驶监测和预警的需求尤为突出，尤其是在公共交通和物流运输领域。本项目旨在研究和开发一个基于智能软件的疲劳驾驶动态监测平台，以提高道路安全性和减少交通事故。

1.2 项目目的

项目的主要目的是设计并实现一个高效、准确的疲劳驾驶监测系统。该系统应能够实时监测驾驶员的面部特征，分析疲劳状态，并在必要时发出预警，以防止可能由疲劳驾驶引起的交通事故。此外，系统还需提供一个用户友好的界面，使得司机和管理人员能够轻松地操作和获取相关信息。

1.3 项目范围

本项目的研究与开发范围限定于疲劳驾驶监测系统的设计、实现和测试，需要研究的技术包括计算机视觉、人脸识别算法、实时数据处理等。系统应能在常见的操作系统和网络环境下稳定运行。

2 项目成员职务及任务分解

2.1 成员角色概述

项目团队由五名成员组成，分别担任不同的关键角色，确保项目的顺利进行。

陈昊天：作为项目经理、产品经理和算法工程师，负责整体项目规划与管理、产品需求分析和疲劳检测算法的实现。他主导了项目管理计划的制定、产品需求文档的撰写，并设计实现了疲劳检测的前后端功能。

戴建雨和周佳羽：身兼 UI 设计师和研发工程师，分别负责司机和管理员用户界面的设计与实现。她们的工作包括 UI 设计、前后端功能的实现和测试。

严贺阳阳和石查分：作为系统架构师和测试研发工程师，她们共同设计了系统总体结构，并执行了系统测试。她们定义了系统架构、编写了测试计划和测试用例，并记录了测试中发现的问题。

2.2 成员职务及任务分解表

成员姓名	职务	主要职责	具体任务
陈昊天	项目经理 产品经理 算法工程师	整体项目规划与管理 产品需求分析 设计疲劳检测用户界面 实现疲劳检测前后端功能	制定项目管理计划 撰写产品需求文档 设计疲劳检测界面原型 编码实现疲劳检测算法及其前后端集成
戴建雨	UI 设计师 研发工程师	设计司机用户界面 实现司机前后端功能	创建司机界面 UI 设计 实现司机使用的前后端功能 进行前后端的接口测试和优化
周佳羽	UI 设计师 研发工程师	设计管理员用户界面 实现管理员前后端功能	创建管理员界面 UI 设计 实现管理员使用的前后端功能 进行前后端的接口测试和优化
严贺阳阳	系统架构师 测试研发工程师	设计系统总体结构 执行测试并记录缺陷	协助设计系统架构 编写测试计划和测试用例 执行系统测试，记录并报告问题
石查分	系统架构师 测试研发工程师	设计系统总体结构 执行测试并记录缺陷	定义系统架构和技术栈 协助编写测试计划和测试用例 执行系统测试，记录并报告问题

表 2-1 成员职务及任务分解

3 项目的开发计划

3.1 开发计划概述

项目的开发计划详细规划了从需求分析到系统部署的全过程。在启动阶段，团队将进行需求分析和技术选型，确立面部特征检测算法和系统基本功能。随后的计划阶段，将设计系统架构，包括面部特征检测、实时监测、车辆定位、车辆

管理和司机管理等模块。开发阶段重点在于各模块功能的实现和界面开发，涵盖面部特征检测、司机界面、管理员界面和登录界面的开发。收尾阶段，则聚焦于系统集成、测试和部署，确保系统的稳定性、性能和安全性。

3.2 项目开发计划表

项目开发计划表					
执行阶段	主要工作内容	具体动作	开始时间	完成时间	主要参与人员
启动阶段	1.需求分析和定义	1.分析司机的需求和期望	12.11	12.11	戴建雨
		2.分析管理员的需求和期望	12.11	12.11	周佳羽
		3.确定面部特征检测算法	12.11	12.11	陈昊天
		4.确定系统的基本功能	12.11	12.11	严贺阳阳
	2.技术选型	1.选择计算机视觉库和工具	12.11	12.12	陈昊天
		2.确定开发平台	12.11	12.12	石查分
计划阶段	3.系统架构设计	1.面部特征检测模块架构	12.12	12.14	陈昊天
		2.实时监测模块架构	12.12	12.14	陈昊天
		3.车辆定位模块架构	12.12	12.14	戴建雨
		4.车辆管理模块架构	12.12	12.14	周佳羽
		5.司机管理模块架构	12.12	12.14	戴建雨
		6.制定警告模块架构	12.12	12.14	周佳羽
		7.确定模块之间的通信和数据流程	12.12	12.14	陈昊天
开发阶段	4.面部特征检测模块开发	1.开发从摄像头捕获面部特征并判断是否疲劳的算法	12.14	12.15	陈昊天
		2.测试并优化算法的性能和准确度	12.15	12.16	陈昊天

	5.司机界面开发	1.实现主界面	12.14	12.16	戴建雨
		2.实现个人中心界面	12.14	12.16	戴建雨
		3.实现监控中心界面	12.14	12.16	戴建雨
		4.实现实时监控	12.14	12.16	陈昊天
		5.实现警告功能	12.14	12.16	陈昊天
	6.管理员界面开发	1.实现主界面	12.14	12.16	周佳羽
		2.实现基础信息界面	12.14	12.16	周佳羽
		3.实现监控中心界面	12.14	12.16	周佳羽
		4.实现视频监控	12.14	12.16	陈昊天
		5.实现实时数据更新	12.14	12.16	陈昊天
	7.登录界面开发	1.实现用户登录	12.14	12.15	周佳羽
		2.实现用户注册	12.14	12.15	戴建雨
收尾阶段	8.系统集成和测试	1.模块整合并进行集成测试	12.17	12.18	严贺阳阳
		2.测试系统的稳定性、性能和安全性	12.17	12.18	石查分
	9.测试和反馈	1.小组内进行测试,收集反馈	12.18	12.19	严贺阳阳
		2.根据反馈进行必要的调整和改进	12.18	12.19	陈昊天
	10.部署和维护	1.部署系统到实际环境中,确保其正常运行	12.18	12.20	陈昊天

表 3-1 项目的开发计划

4 可行性及需求分析

4.1 可行性分析

4.1.1 技术可行性

从技术可行性方面来看，疲劳驾驶动态监测平台是一种基于计算机视觉技术来实现疲劳驾驶监测和报警功能的监控管理平台。它通过摄像头捕捉画面中的人脸图像，然后利用人脸识别算法对图像进行分析和比对，实现疲劳驾驶的判断。同时利用前端技术完成用户界面和良好交互的实现。

1. 疲劳驾驶识别技术

人脸识别技术已经得到广泛应用和研究，并在疲劳驾驶检测领域取得了较好的效果。

选用 HOG 结合 SVM 分类算法进行面部检测和关键特征点数据提取，该算法已经被广泛验证并具有较高的准确性和效率。采用 Dlib 库构建面部表情对照模型，可以有效地识别驾驶员的面部表情，判断其是否处于疲劳状态。结合 PERCLOS 指标来判断驾驶员的疲劳驾驶状态，该指标已经在疲劳驾驶研究中被广泛使用，具有一定的可靠性和准确性。

2. 用户界面和用户体验

系统拥有用户友好的界面，使得司机能够轻松理解和响应系统的提示。利用直观的图形和视频界面，管理员能够实时监测多个司机的状态。

应用前后端分离的架构提高系统的灵活性和可维护性。前端通过 API 接口与后端进行数据交互，使得团队可以独立开发、测试和部署前端部分。Vue.js 和 Element UI 的组合使得用户界面的设计更加直观、友好，有助于提高用户体验。通过响应式设计和组件化开发，可以更好地适应不同屏幕大小和设备类型。结合采用 Vue.js 的组件生命周期和钩子函数，调用后端接口进行数据获取和更新。

4.1.2 经济可行性

1. 开发成本

从开发成本来看，疲劳驾驶动态监测平台的开发成本主要包括硬件设备、软件开发、数据存储、运行维护和人力成本等方面的投入，这些费用都需纳入到经济可行性的考虑范围内。

2. 平台收益

从平台带来的收益来看，平台后续可以开发订阅功能。通过提供疲劳驾驶监测平台，满足用户对驾驶安全的需求，吸引更多的司机和车队管理者使用该系统，继而直接转化为用户订阅费用的增加，提升平台的盈利能力。此外，平台未来还能提供额外的服务，如数据报告、定期更新、定制化功能等，用户可以选择付费获取这些增值服务。从更长远的角度分析，平台可以联系企业，面向企业客户提供定制化的车队管理解决方案，包括实时监控、数据分析、司机管理等功能。同时在社会上，良好的社会形象可能吸引更多用户和合作伙伴，间接影响平台的盈利能力。

4.1.3 市场可行性

1. 道路安全意识

道路交通事故中疲劳驾驶是一个常见的原因，因此，社会对提高道路安全性的需求日益增加。许多国家和地区对于驾驶员疲劳驾驶的法规要求逐渐提高，强化了对于驾驶员状态监测的需求。具有疲劳驾驶监测功能的平台能够满足人们对道路安全提升的迫切需求，吸引用户关注和参与。提供符合法规标准的疲劳驾驶监测平台能够满足企业和驾驶员对合规性的需求，避免违法行为。

2. 实时监控与通知功能

在疲劳驾驶监测中，实时监控和及时通知对于防止事故的发生至关重要。因此平台提供了实时监控中心和紧急通知功能，确保管理员和司机能够在第一时间获取关键信息。

4.1.4 可行性风险和挑战

1. 面部识别技术

在实际应用中，面部表情的识别和疲劳状态的判断可能会受到多种因素的影响，如光线条件、驾驶员个体差异等，需要进行充分的算法优化和测试。同时，算法的实时性和效率对于驾驶员监测系统至关重要，需要在算法选择上做出权衡。数据隐私和安全问题也需要充分考虑，确保驾驶员的个人信息和数据得到合理的

保护。

2. 平台实现方面

在开发过程中，需要权衡实时性和可扩展性，确保系统在不同场景下都能够快速响应。对网络连接的依赖性需要谨慎，尤其是在移动应用场景下。

在使用平台时，界面需要简单直观，避免分散驾驶员的注意力。在管理员界面中，需要提供足够的信息，同时避免信息过载，确保易用性。

系统在各种驾驶场景和不同车辆中的性能需要详细测试，以确保广泛的适用性。针对紧急情况的处理机制，如系统故障或网络中断，需要仔细规划和实施。

4.2 需求分析

4.2.1 功能需求

1. 用户注册与登录

提供司机和管理员注册账户的功能，并支持登录验证。如果账号判断为司机，则跳转到司机界面，反之则跳转到管理员界面。

2. 实时面部监测

实现对驾驶员面部的实时监测，检测可能的疲劳驾驶迹象。一旦监测到司机有疲劳驾驶的状态，就发出警报提醒。

3. 位置显示

在主界面显示地图上的当前位置，提供实时位置信息。

4. 个人中心

对司机提供个人中心，允许查看基础信息包括账号信息、疲劳记录和交规必知等。

5. 监控中心

对管理员提供监控中心，允许管理员实时查看驾驶员的状态。监控中心应包括多个监控画面，以便管理员全面地监控司机们的状态。

6.系统管理

对管理员提供系统管理功能，包括用户和车辆的查看与管理。

7.数据分析报告

提供疲劳驾驶数据的分析报告，帮助管理员了解驾驶员的驾驶行为。

4.2.2 数据需求

1.面部监测数据

从摄像头中收集驾驶员面部监测数据，包括面部特征、表情等传入后端进行疲劳驾驶判断。

2.位置信息

获取司机的实时位置信息，以显示在地图上。

3.用户信息

存储司机和管理员的基础信息，包括用户名、密码、车辆信息等。

4.交规必知数据

提供交规必知模块所需的法规和安全知识数据。

4.2.3 性能需求

1.实时性

面部监测需要在实时性较高的情况下进行，确保对疲劳驾驶的及时响应。为了提高速度，可以利用 GPU 加速、优化算法实现等方法来提升处理效率。

2.准确性

面部监测和疲劳状态判断的准确性是关键，确保误判率低。提高准确率可以通过使用高质量的摄像头捕捉清晰的人脸图像，采用高性能的人脸识别算法等手段实现。

3.可拓展性

平台应具有可拓展性，以适应未来功能的扩展和新的用户需求。另外平台应

能适配多种摄像头设备和操作系统，以满足不同场景和用户需求。

4. 并发性

确保系统能够处理多用户同时访问和监测的情况，保证系统的高并发性能。

5. 隐私与安全性

系统应采取有效的措施保护个人隐私信息，确保人脸图像和人脸特征的安全性，避免数据泄露和滥用。

5 系统设计

5.1 系统设计概述

该系统是一个基于 Vue.js 和 Flask 的疲劳驾驶监控系统，主要功能包括用户登录、注册、车辆监控、疲劳检测、数据统计、实时视频监控等。系统前端使用 Vue.js 框架搭建，后端使用 Flask 框架。系统通过 Websocket 进行实时通信，使用高德地图 API 进行地图展示和车辆定位。

5.1.1 软件流程图

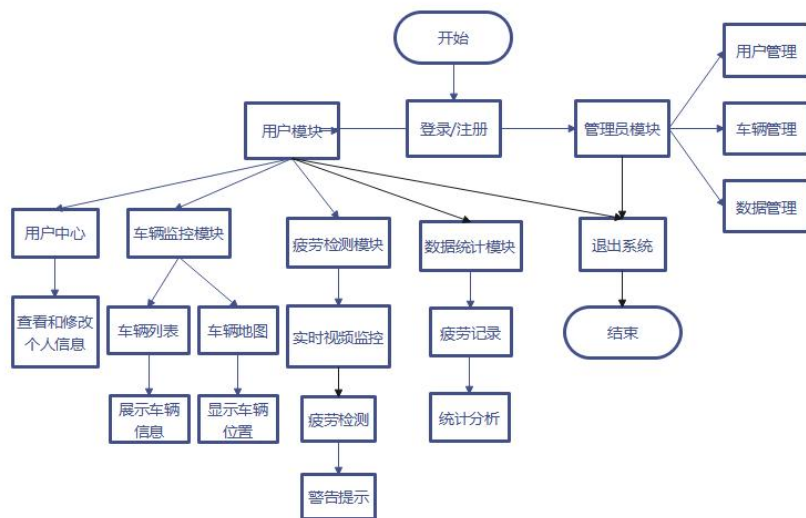


图 5-1 疲劳驾驶监控系统流程图

5.1.2 系统架构图

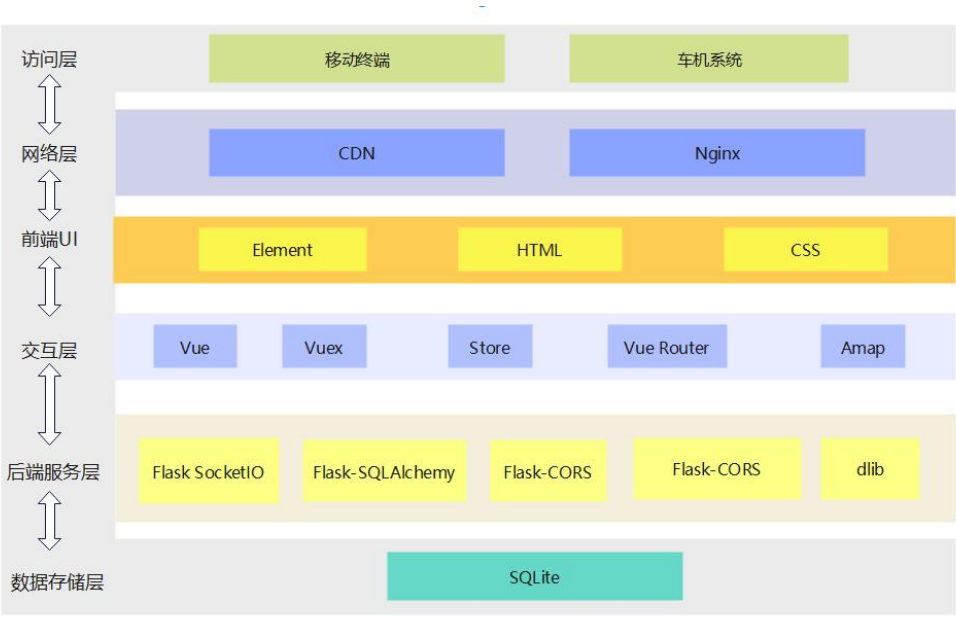


图 5-2 疲劳驾驶监控系统系统架构图

5.2 前端设计

前端设计是整个系统中至关重要的一部分，负责用户界面和用户体验的开发。

1. Vue.js 框架和组件化开发

在前端开发中，选择了 Vue.js 作为主要的开发框架。Vue.js 以其简洁灵活的特性而闻名，采用组件化开发的方式。组件化开发使得代码更易维护和扩展，同时提高了代码的复用性。每个功能或界面元素都可以被封装成一个独立的组件，这样整个应用的结构更清晰。

2. Element UI 库的使用

为了更高效地进行界面布局和交互设计，采用了 Element UI 库提供的丰富组件。Element UI 提供了一套美观而实用的界面元素，包括表单、表格、对话框等，使得开发者可以更专注于业务逻辑的实现而不必过多关注样式和布局。

3. Vue Router 和单页面应用

通过 vue-router 进行页面路由管理，实现了单页面应用。这种方式通过动态加载页面组件，避免了整页刷新，提升了用户体验。同时，vue-router 的路由配置使得页面间的跳转和导航更加简便。

4. Vuex 状态管理

使用 Vuex 进行状态管理，通过一个全局的状态树来管理应用中的共享数据。这样，不同组件之间的数据传递和状态同步变得更加容易。Vuex 的单一状态树和状态变更的追踪机制有助于更好地维护和调试应用的状态。

5. Axios 进行 HTTP 请求

通过 Axios 库进行 HTTP 请求，与后端 API 进行数据交互。Axios 是一个强大的 HTTP 客户端，支持异步请求和 Promise API，使得与后端的通信更加高效和可靠。

6. vue-amap 集成高德地图

集成了 vue-amap，这是一个 Vue.js 地图组件，用于展示高德地图并实现车辆定位功能。这为用户提供了直观的地理信息展示，并与其他功能集成，如车辆监控。

7. Socket.IO 进行 Websocket 通信

使用 Socket.IO 实现了 WebSocket 通信，实时接收后端推送的数据。这使得系统能够在需要实时性的场景下，如实时视频监控和疲劳检测，保持与后端的即时连接，实现实时数据传输和更新。

5.3 后端设计

后端设计在系统的架构中扮演着关键的角色，负责处理业务逻辑、数据存储和与前端的交互。

1. Flask 框架和 RESTful API

后端选择了 Flask 作为主要的框架，它是一个轻量级而灵活的 Python Web 框架。通过 Flask，搭建了 RESTful API 接口，使得前后端之间的数据交换更加简便和规范。RESTful 设计风格提供了一种清晰、可扩展且易于理解的方式来组织 API，使得系统的通信变得高效而易于维护。

2. Flask-SocketIO 进行 WebSocket 通信

利用 Flask-SocketIO，实现了 WebSocket 通信，与前端建立了实时的双向通道。这使得系统能够实时地推送数据给前端，例如实时车辆位置、驾驶状态等信息。WebSocket 的实时性和高效性为系统的实时监控和通知功能提供了强有力的支持。

3. Flask-SQLAlchemy 进行数据库操作

使用 Flask-SQLAlchemy 进行数据库操作，这是 Flask 的一个 SQLAlchemy 扩展，简化了与数据库的交互。通过它，系统可以轻松地连接和操作数据库，存储用户信息、车辆信息以及疲劳记录等关键数据。SQLAlchemy 的 ORM（对象关系映射）模式提供了一种方便的方式来处理数据库交互，使得数据的管理和查询更为灵活。

4. OpenCV 和 dlib 进行人脸特征检测

利用 OpenCV 和 dlib 库，实现了人脸特征检测，以用于疲劳驾驶检测功能。这一功能通过分析驾驶员的面部表情和特征，识别出可能的疲劳迹象，提醒驾驶员采取必要的休息措施。OpenCV 和 dlib 为图像处理和机器学习任务提供了丰富的工具和算法。

5. CORS 解决跨域请求问题

使用 CORS 机制解决了跨域请求的问题。这是一种安全机制，允许在浏览器中运行的 Web 应用程序从不同的域访问其资源。通过配置合适的 CORS 头，确保前端能够安全地发起对后端的跨域请求，确保系统的安全性和可用性。

5.4 数据库设计

5.4.1 数据库设计概述

数据库设计涉及车辆、用户、和疲劳记录。Car 模型代表了车辆的信息。每辆车都有一个唯一的 id 作为主键, 以及 plates 字段存储车牌号, 这个字段是唯一的且不允许为空。车辆的类型和所在城市分别由 style 和 city 字段表示。还有一个 status 字段, 用来描述车辆的当前状态。

User 模型代表了系统的用户。每个用户都有一个唯一的 id, 以及 name 和 phone 字段, 后者是唯一的且不允许为空。用户的地址和所在城市分别通过 address 和 city 字段存储。

FatigueRecord 模型用来记录用户的疲劳情况。这个模型包含一个主键 id, 两个外键 user_id 和 car_id 分别指向 User 和 Car 模型的 id 字段, 确立了疲劳记录与特定用户和车辆的联系。record_time 字段记录了疲劳发生的时间, duration 字段表示疲劳持续的时间。

5.4.2 数据库 ER 图

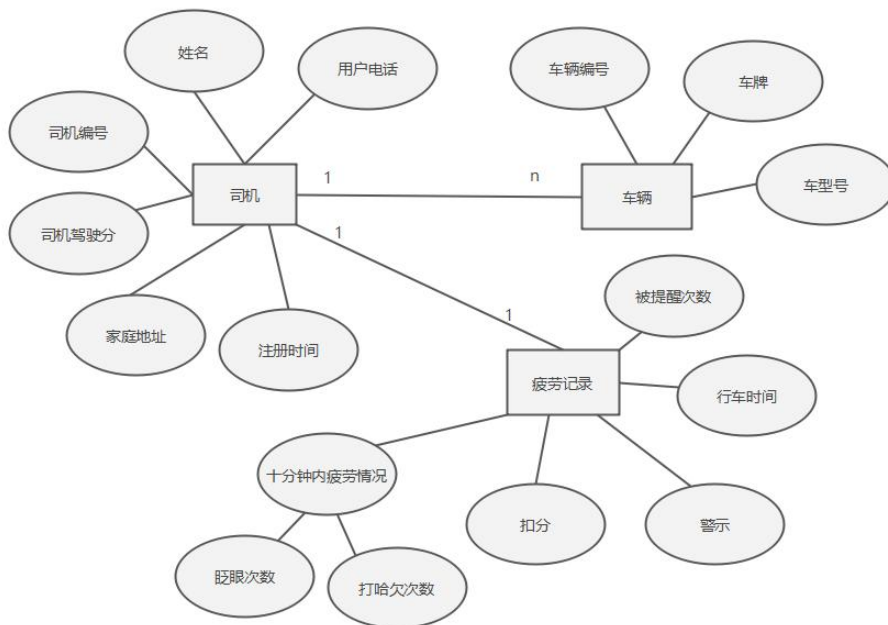


图 5-3 疲劳驾驶系统数据库 ER 图

5.5 安全设计

1. 使用 HTTPS 进行数据传输

HTTPS 提供了加密的数据传输通道，通过使用 SSL/TLS 协议对数据进行加密，确保在用户和服务器之间传输的数据是加密的。这可以防止中间人攻击和数据窃听，提升数据传输的安全性。

2. 加密处理

在数据库存储和数据传输过程中使用加密是关键。这包括对敏感数据（如用户信息、驾驶记录等）进行加密存储，使用强加密算法确保数据在存储和传输过程中的安全性。合适的加密措施可以保护数据不被未经授权的访问者获取或窃取。

3. 对输入数据进行校验

在系统接收和处理用户输入数据时，进行有效的数据验证和过滤是至关重要的。通过实施输入验证，可以防止恶意用户利用输入字段执行 SQL 注入、XSS（跨站脚本攻击）等攻击。验证输入数据的格式、长度、范围和类型，确保它们符合预期的标准，防止恶意数据进入系统。

6 模块设计

模块设计是疲劳驾驶动态监测平台的核心组成部分，通过合理的模块划分和设计，确保系统的可维护性、可拓展性和高效性。本节将对平台的主要模块进行介绍，包括用户模块、车辆监控模块、疲劳监测模块、数据统计模块、系统管理模块以及数据通信模块。

6.1 模块汇总

6.1.1 模块汇总表

模块名称	功能简述
用户子系统	
登录	允许已注册用户输入用户名和密码登录系统
注册	允许新用户创建账户，提供必要信息以完成注册过程
个人中心	允许用户查看和管理个人信息
车辆监控子系统	
车辆列表	展示所有车辆的信息，包括车辆位置、状态等
车辆地图	在地图上显示车辆的位置，可以查看车辆的实时状态
疲劳检测子系统	
实时视频监控	通过摄像头捕捉驾驶员的面部图像
疲劳检测	分析驾驶员的面部特征，如眼睛闭合度、嘴巴张开程度等，检测疲劳状态
警告提示	当检测到疲劳驾驶行为时，系统会发出警告提示
数据统计子系统	
疲劳记录	记录驾驶员的疲劳驾驶行为，包括时间、持续时长等
统计分析	对疲劳驾驶数据进行统计分析，生成报表
系统管理子系统	
用户管理	管理员可以管理系统中的用户账号
车辆管理	管理员可以管理系统中的车辆信息
数据管理	管理员可以管理系统中的各类数据
数据通信子系统	
全双工通信	实现前后端的实时通信，传递疲劳检测结果和警告信息
负载均衡器	对入站的请求流量和视频传输流量进行负载均衡
国际网络优化	采用三大运营商的优化线路确保全球访问的可靠性

表 6-1 模块汇总表

6.1.2 模块框架图

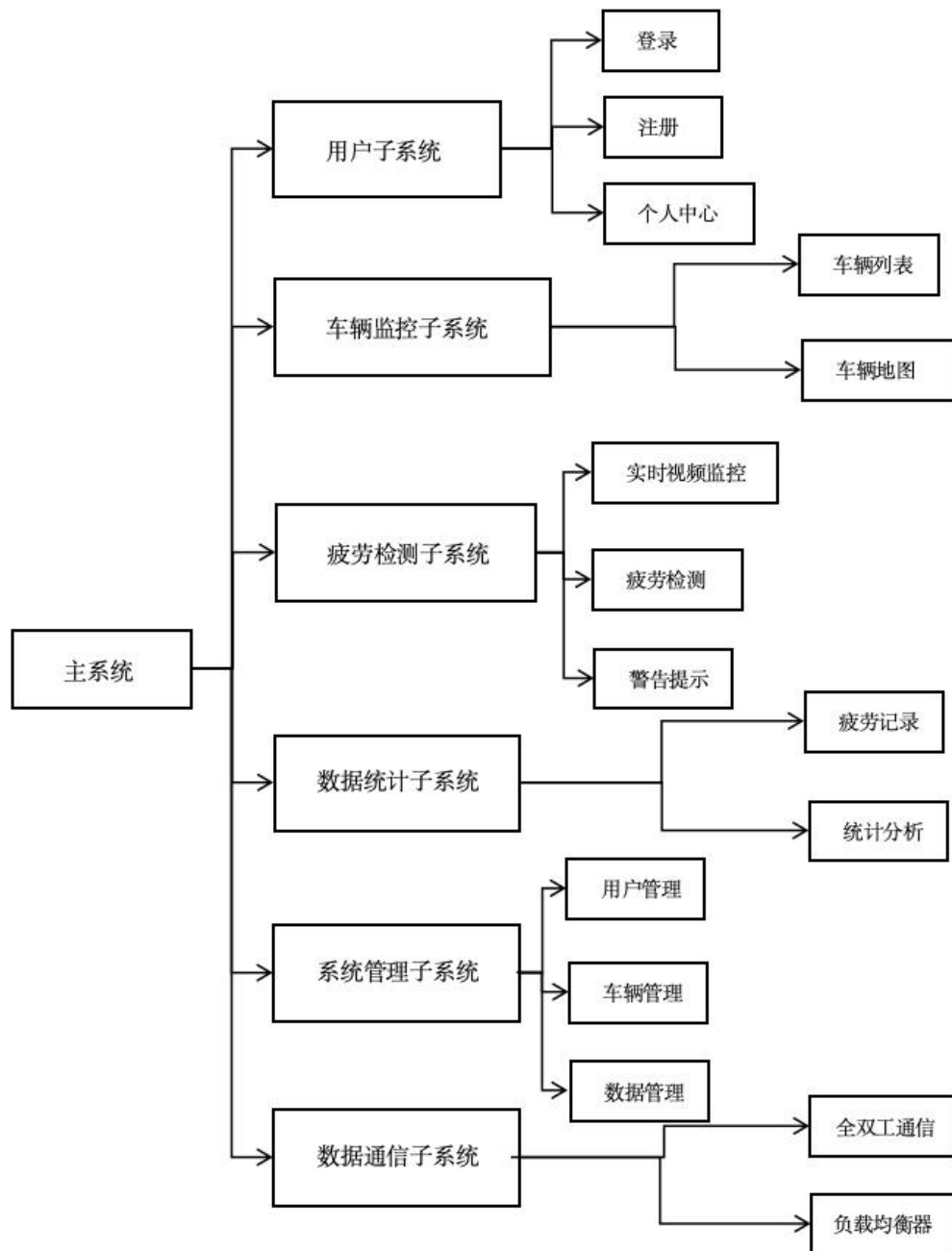


图 6-1 模块框架图

6.2 用户子系统的模块设计

6.2.1 登录模块


模块名称	用户登录
功能描述	允许已注册用户输入用户名和密码登录系统
主要算法实现	<p>前端发送登录请求，请求方式为 POST，发送的数据为 JSON 格式，包含用户的用户名和密码。</p> <pre> export const accountLogin = (Name, Pwd) => { return request({ url: '/Account/AccountLogin', method: 'post', headers: { 'Content-Type': 'application/json', }, data: { Name, Pwd } }) } </pre> <p>后端接口负责验证用户提供的用户名和密码，并根据验证结果返回相应的响应。如果用户名和密码匹配，返回登录成功的响应，否则返回错误信息</p> <pre> name = request.json.get("Name") pwd = request.json.get("Pwd") print(f"{name}, {pwd}") for login in users: if login["userAccount"] == name and login["userPassword"] == pwd: carPlates = ([car["carPlates"] for car in cars if car["userId"] == login["userId"] and login["type"] == 1],) response = { "code": 20000, "message": "登录成功", "data": { "token": </pre>

	<pre>"fake-jwt-token-for-demo", "type": login["type"], "sessionId": login["userId"], "userName": login["userName"], "userAccount": login["userAccount"], "carPlates": carPlates[0], }, } return jsonify(response) if login["userAccount"] == name and login["userPassword"] != pwd: response = {"code": 4002, "message": " 管理员用户名对应密码错误"} return jsonify(response) response = {"code": 4001, "message": "本管理员用 户名不存在"} return jsonify(response)</pre>
界面展示	

6.2.2 注册模块

模块名称	用户注册
功能描述	允许新用户创建账户，提供必要信息以完成注册过程
主要算法实	前端封装了一个向后端发送用户注册请求的过程，包括请求的

现	<p>目标路径、HTTP 方法、请求头和请求体 (包含用户名和密码)</p> <pre> export const register = (admin, password) => { return request({ url: '/Account/AccountRegister', method: 'post', headers: { 'Content-Type': 'application/json', }, data: { admin, password } }) } </pre> <p>后端接口接收前端注册请求, 验证用户名和密码是否合法, 如果合法则生成一个新用户并返回注册成功的响应。如果注册过程中出现异常, 会返回注册失败的响应</p> <pre> @app.route("/Account/AccountRegister", methods=["POST"]) def admin_register(): data = request.get_json() print("data", data) name = data.get("admin") password = data.get("password") type = 1 try: if name != "" and password != "": userId = len(users) new_user = { "userId": userId, "userName": "", "userPhone": "", "userTime": datetime.now().date().strftime("%Y-%m-%d"), "userScore": 12, "userAddress": "", "userAccount": name, "userPassword": password, "type": 1, } users.append(new_user) response = { </pre>
---	---

	<pre> "code": 20000, "message": "register success", "data": { "token": "fake-jwt-token-for-demo", }, }, } return jsonify(response) except Exception as e: print("register failed: ", e) response = {"code": 4001, "message": "register failed"} return jsonify(response) </pre>
界面展示	

6.2.3 个人中心模块

模块名称	个人中心
功能描述	允许用户查看和管理个人信息
主要算法实现	<p>通过调用 load 方法，使用 userInfo 函数向后端发起请求，获取用户信息和事件数据，并更新本地状态</p> <p>在组件挂载后，通过 WebSocket 连接监听后端推送的事件数据 (socket.on('response', ...))，并在收到事件数据时更新本地事件状态</p> <p>在组件销毁前，通过 beforeDestroy 钩子断开 WebSocket 连接。</p> <pre> mounted() { this.load(); </pre>


```

    this.socket.on('response', (data) => {
        this.events = data.events
    });
},
methods: {
    async load() {
        const response = await
userInfo(this.$cookies.get("usersessionid"));
        const { code, res, events } = response.data;
        this.user = res
        this.events = events
        this.events = this.events.filter(event =>
event["userId"] ==
this.$cookies.get('usersessionid'));
    },
    edit() {
        this.dialogFormVisible = true;
    },
    handleClose() {
        this.$refs.updateuser.resetFields()
    },
    submit() {
        this.$refs.updateuser.validate(async valid =>
        {
            if (!valid) return
            // ... 发起修改信息的数据申请 ...
        })
    },
    watch: {
        // ... 深层监听 ...
    },
    beforeDestroy() {
        this.socket.disconnect();
    }
}

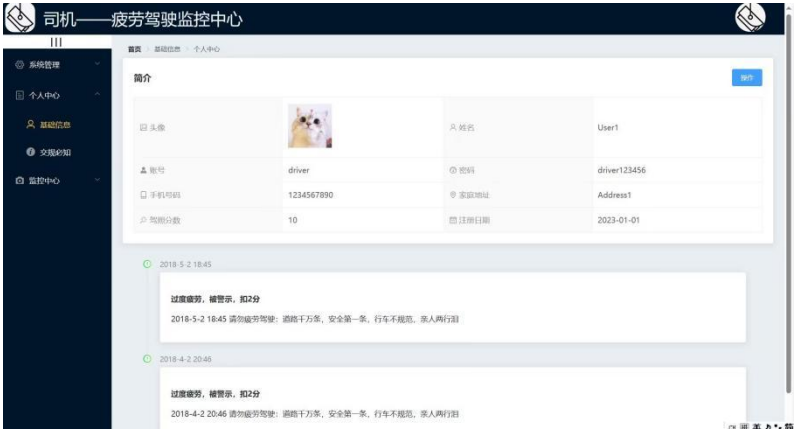
```

基于 Flask 框架的后端接口接收前端传递的用户 ID 参数, 根据 ID 筛选出符合条件的用户信息, 并将结果以 JSON 格式返回给前端

```

@app.route("/User/userInfo", methods=["GET"])
def userInfo():
    target = int(request.args.get("userId"))
    print(target)
    res_user = [

```

	<pre>user for user in users if user["userId"] == target and user["type"] == 1] return jsonify({"code": 20000, "res": res_user[0], "events": events})</pre>
界面展示	

6.3 车辆监控子系统的模块设计

6.3.1 车辆列表模块

模块名称	车辆列表
功能描述	展示所有车辆的信息，包括车辆位置、状态等
主要算法实现	<p>前端在组件创建时，通过 myqueryCarList 方法查询所有车辆的信息，通过 getCarCityAndCount 方法获取车辆城市和数量的信息。</p> <p>通过异步方法 getCarCityAndCount，调用后端接口获取车辆城市和数量的信息，将结果存储在 carCityAndCount 变量中。</p> <p>通过异步方法 myqueryCarList，调用后端接口查询车辆列表信息，更新 carList、current 和 total 变量，分别表示车辆列表数据、当前页码和总条数。</p> <p>提供了 resetForm 方法，用于重置筛选条件，并调用 myqueryCarList 方法重新查询数据。</p> <pre>created () {</pre>

```

    this.myqueryCarList()
    this.getCarCityAndCount()
  },
  methods: {
    handleSizeChange (val) {
      // ... 查询数据 ...
    },
    handleCurrentChange (val) {
      this.current = val
      this.myqueryCarList()
    },
    async getCarCityAndCount () {
      const { data } = await findCarCityAndCount()
      console.log(data)
      this.carCityAndCount =
        data.data.carCityAndCount
    },
    async myqueryCarList () {
      const { data } = await
        queryCarList(this.current, this.pageSize,
          this.formInline.city, this.formInline.plate,
          this.formInline.style)
      this.carList = data.data.records
      this.total = data.data.total
    },
    resetForm () {
      // ... 更新表格 ...
    }
  }
}

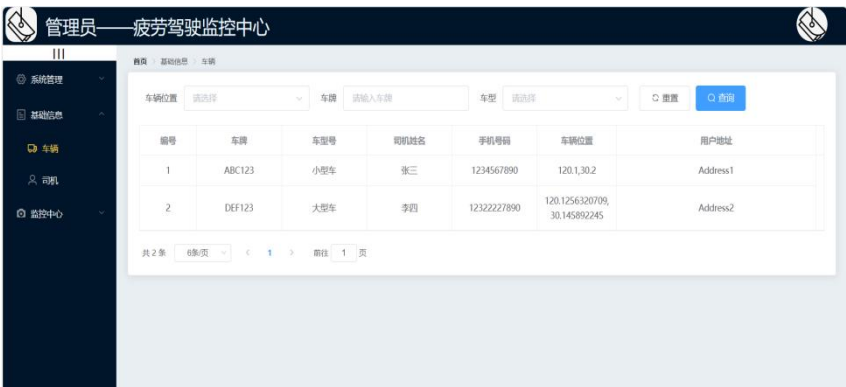
```

后端接口接收前端的查询参数，根据参数进行车辆信息的筛选，并将筛选结果返回给前端

```

@app.route("/Car/queryCarList", methods=["GET"])
def query_car_list():
    city = request.args.get("carTude")
    plates = request.args.get("carPlates")
    style = request.args.get("carStyle")
    filtered_car = cars
    if city:
        filtered_car = [item for item in filtered_car
            if item["carTude"] == city]
    if plates:
        filtered_car = [

```

	<pre> item for item in filtered_car if item["carPlates"].find(plates) != -1] if style: filtered_car = [item for item in filtered_car if item["carStyle"] == style] print("filtered_car", filtered_car) return jsonify({"code": 20000, "data": {"records": filtered_car, "total": len(cars)}}) </pre>
界面展示	


6.3.2 车辆地图模块

模块名称	车辆地图
功能描述	在地图上显示车辆的位置，可以查看车辆的实时状态
主要算法实现	<p>在组件创建时调用 <code>getCarsAndUser</code> 方法, 获取车辆和用户的信息。<code>loadMap</code> 方法作为地图初始化完成后的回调，调用一些初始化地图时需要的方法，包括获取未使用车辆、获取正在使用车辆和定时获取车辆距离。使用 <code>findStoppedCars</code> 和 <code>findStartedCars</code> 方法获取未使用和正在使用的车辆信息。</p> <p>处理获取到的车辆信息，设置图标、文本信息等，并通过回调传递给地图组件。使用 <code>updateDiatance</code> 方法获取车辆距离信息，定时更新显示距离。</p> <pre> export default { name: 'UserHost', </pre>

```

components: { Amap },
created() {
  this.getCarsAndUser()
},
methods: {
  callbackComponent(params) {
    params.function &&
    this[params.function](params.data)
  },
  async getCarsAndUser() {
    const { data } = await findCarsAndUser()
    this.startedCarNum = data.data.startedCarNum
    this.stoppedCarNum = data.data.stoppedCarNum
    this.carNum = this.startedCarNum +
this.stoppedCarNum
    this.userNum = data.data.userNum
  },
  loadMap() {
    // ... 加载地图 ...
  },
  async getStartedCar() {
    const { data } = await findStartedCars()
    data.data.startedCarList.forEach(item => {
      const bgColor = item.userName ===
this.$cookies.get("userName") ? 'white' : '#42b983'
      const status = item.userNow === 0 ? '疲劳驾
驶' : '状态正常'
      item.text = this.generateCarText(item,
'red', bgColor, status)
      item.offset = [10, -40]
      item.position = item.carTude.split(',')
    })
    this.$refs.map.findstartCarsData(data)
  },
  async getDistance() {
    // ... 获取车辆信息 ...
  },
  generateCarText(item, textColor) {
    // ... 生成车辆展示信息 ...
  },
  setIntervalTasks(tasks, interval) {
    tasks.forEach(task => {
      this.timer = setInterval(task, interval)
    })
  }
}


```

	<pre> } }, beforeDestroy: function () { clearInterval(this.timer) } } </pre>
界面展示	

6.4 疲劳检测子系统的模块设计

6.4.1 实时视频监控模块

模块名称	实时视频监控
功能描述	通过摄像头捕捉驾驶员的面部图像
主要算法实现	<p>前端代码通过 navigator.mediaDevices.getUserMedia 获取用户的摄像头流。利用 Canvas 元素将摄像头捕捉到的视频流进行处理，定时获取视频帧，转换成 Base64 编码的图像数据，并通过 WebSocket 发送给后端进行疲劳检测。</p> <pre> export default { name: 'UserFatigueDetection', data() { return { videoElement: null, userId: '', socket: io(process.env.VUE_APP_SOCKET_URL), }; }, mounted() { this.userId = this.generateUserId(); } } </pre>

	<pre> this.videoElement = this.\$refs.videoElement; navigator.mediaDevices?.getUserMedia({ video: true }).then((stream) => { this.videoElement.srcObject = stream; this.videoElement.play(); const canvas = document.createElement('canvas'); const context = canvas.getContext('2d'); const maxWidth = 300; setInterval(() => { context.drawImage(this.videoElement, 0, 0, canvas.width, canvas.height); const dataURL = canvas.toDataURL('image/jpeg'); this.socket.emit('frame', { image: dataURL.split(',')[1], userId: this.userId }); }, 200); }); }); }, beforeDestroy() { this.socket.disconnect(); } }; </pre>										
界面展示	 <table border="1"> <thead> <tr> <th>项目</th> <th>数据</th> </tr> </thead> <tbody> <tr> <td>睡眠计数</td> <td>0</td> </tr> <tr> <td>眼部状态</td> <td>睡眠</td> </tr> <tr> <td>打哈欠计数</td> <td>1</td> </tr> <tr> <td>眼部状态</td> <td>打哈欠</td> </tr> </tbody> </table>	项目	数据	睡眠计数	0	眼部状态	睡眠	打哈欠计数	1	眼部状态	打哈欠
项目	数据										
睡眠计数	0										
眼部状态	睡眠										
打哈欠计数	1										
眼部状态	打哈欠										

6.4.2 疲劳检测模块

模块名称	疲劳检测
------	------

功能描述	分析驾驶员的面部特征，如眼睛闭合度、嘴巴张开程度等，检测疲劳状态
主要算法实现	<p>后端接口接收前端的摄像头流, 加载 Dlib 的人脸检测器和形状预测器, 计算纵横比。根据预设的阈值和连续帧计数来判断用户是否在眯眼或打哈欠, 使用计数器跟踪连续闭眼和连续打哈欠的次数</p> <pre> def eye_aspect_ratio(eye): # ... 计算眼睛的纵横比 ... def mouth_aspect_ratio(mouth): # ... 计算嘴巴的纵横比 ... @socketio.on("frame", namespace="/ws") def handle_frame(data): user_id = data["userId"] # ... 使用用户特定的计数器 ... img_data = base64.b64decode(data["image"]) nparr = np.frombuffer(img_data, np.uint8) frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR) gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) faces = detector(gray) for face in faces: (x, y, w, h) = face_utils.rect_to_bb(face) cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2) shape = predictor(gray, face) shape = face_utils.shape_to_np(shape) for x, y in shape: cv2.circle(frame, (x, y), 1, (0, 0, 255), -1) left_eye = shape[42:48] right_eye = shape[36:42] left_eye = eye_aspect_ratio(left_eye) right_eye = eye_aspect_ratio(right_eye) ear = (left_eye + right_eye) / 2.0 if ear < EYE_AR_THRESH: EYE_COUNTER += 1 eye_text = "眯眼" else: if EYE_COUNTER >= EYE_AR_CONSEC_FRAMES: EYE_TOTAL += 1 EYE_COUNTER = 0 mouth = shape[48:68] </pre>

	<pre> mar = mouth_aspect_ratio(mouth) # ... 检查嘴巴纵横比是否大于阈值 ... _, buffer = cv2.imencode(".jpg", frame) response = base64.b64encode(buffer).decode("utf-8") # ... 更新用户特定的计数器 ... emit(# ... 传递视频帧 ...) </pre>
界面展示	

6.4.3 警告提示模块

模块名称	警告提示
功能描述	当检测到疲劳驾驶行为时，系统会发出警告提示
主要算法实现	<p>当检测到眼部或嘴部的状态为疲劳时，显示警告卡片，并播放警告声音</p> <pre> export default { name: 'UserFatigueDetection', data() { return { showAlert: false, }; }, methods: { playAlertSound() { const audio = new Audio('/audio/remind.mp3'); audio.play(); }, generateUserId() { return Math.random().toString(36).substring(2, </pre>

	<pre> 15) + Math.random().toString(36).substring(2, 15); } }, }, this.socket.on('response', (data) => { this.imageSrc = 'data:image/jpeg;base64,' + data.data; this.eye_count = data.eye_count; this.eye_text = data.eye_text; this.mouth_count = data.mouth_count; this.mouth_text = data.mouth_text; this.updateTableData(); if (data.eye_count - this.lastEyeCount >= 1 data.mouth_count - this.lastMouthCount >= 1) { this.showAlert = true; this.playAlertSound(); } this.lastEyeCount = data.eye_count; this.lastMouthCount = data.mouth_count; }); </pre>
界面展示	

6.5 数据统计子系统的模块设计

6.5.1 疲劳记录模块

模块名称	疲劳记录
------	------

功能描述	记录驾驶员的疲劳驾驶行为，包括时间、持续时长等
主要算法实现	<p>后端使用用户特定的计数器记录驾驶员的疲劳驾驶行为计数，通过 emit 将信息与视频帧一起传递给前端</p> <pre> @socketio.on("frame", namespace="/ws") def handle_frame(data): user_id = data["userId"] if user_id not in USERS: USERS[user_id] = { "EYE_COUNTER": 0, "MOUTH_COUNTER": 0, "EYE_TOTAL": 0, "MOUTH_TOTAL": 0, } EYE_COUNTER = USERS[user_id]["EYE_COUNTER"] MOUTH_COUNTER = USERS[user_id]["MOUTH_COUNTER"] EYE_TOTAL = USERS[user_id]["EYE_TOTAL"] MOUTH_TOTAL = USERS[user_id]["MOUTH_TOTAL"] # ... 疲劳检测子模块 ... USERS[user_id]["EYE_COUNTER"] = EYE_COUNTER USERS[user_id]["MOUTH_COUNTER"] = MOUTH_COUNTER USERS[user_id]["EYE_TOTAL"] = EYE_TOTAL USERS[user_id]["MOUTH_TOTAL"] = MOUTH_TOTAL emit("response", { "data": response, "eye_count": EYE_TOTAL, "eye_text": eye_text, "mouth_count": MOUTH_TOTAL, "mouth_text": mouth_text, },) </pre>

6.5.2 统计分析模块

模块名称	统计分析
功能描述	对疲劳驾驶数据进行统计分析，生成报表
主要算法	在前端中 getUserTired 方法通过调用 findUserTired 接口来查询用户疲劳驾驶信息，更新 userList、pageSize、total 和 current

实现

```

created() {
  this.getUserTired()
  this.userList = this.userList.filter(user =>
    user.type !== 0);
},
mounted() {
  this.socket.on('response', (data) => {
    this.userList = data.tired
  });
},
methods: {
  async getUserTired() {
    const { data } = await findUserTired(this.current,
    this.pageSize)
    console.log(data.data.UserAll)
    this.total = data.data.total
    this.userList = data.data.UserAll
    console.log('current:' + data.data.current)
    console.log('total:' + data.data.total)
  },
  handleSizeChange(val) {
    console.log(`每页 ${val} 条`)
    this.pageSize = val
    this.getUserTired()
  },
  handleCurrentChange(val) {
    console.log(`当前页: ${val}`)
    this.current = val
    this.getUserTired()
  },
}
}

```

后端处理了前端对获取用户疲劳驾驶信息的请求, 并返回相应的数据, 其中包括用户信息列表和总用户数量

```

@app.route("/Head/findUserTired", methods=["GET"])
def find_user_tired():
    return jsonify({"code": 20000, "data": {"UserAll":
    tired, "total": len(tired)}})

```



6.6 系统管理子系统的模块设计

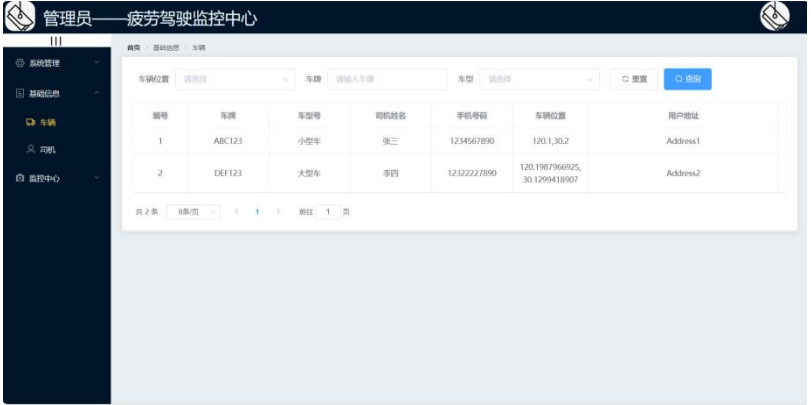
6.6.1 用户管理模块

模块名称	用户管理
功能描述	管理员可以管理系统中的用户账号
主要算法实现	<p>在前端中通过 <code>getUserList</code> 方法：通过调用后端接口 <code>findUserList</code> 获取用户司机信息，根据分页信息更新 <code>userList</code>、<code>total</code> 等数据；<code>editUserInfo</code> 方法：编辑司机信息并提交，通过调用后端接口 <code>editUser</code> 完成。验证通过后，更新司机列表，关闭编辑对话框，显示成功消息；<code>removeUserById</code> 方法：弹出确认对话框，确认后调用后端接口 <code>deleteUser</code> 删除司机信息。删除成功后，更新司机列表，显示成功消息。</p> <pre>methods: { handleSizeChange (val) { // ... 重新从后台查询数据 ... }, handleCurrentChange (val) { console.log(`当前页: \${val}`) this.current = val this.getUserList() }, async getUserList () { // ... 返回对应分页查询的用户司机信息 ... }, async showEditDialog (userId) {</pre>

	<pre> // ... 展示编辑用户的对话框 ... }, editDialogClosed () { // ... 监听修改司机对话框的关闭事件 ... }, // 修改司机信息并提交 editUserInfo () { this.\$refs.editFormRef.validate(async valid => { if (!valid) return // 发起修改信息的数据申请 const { data } = await editUser(this.editForm.userId, this.editForm.userScore, this.reason) console.log(data.data) if (data.code !== 20000) { return this.\$message.error('更新司机信息失败 ') } this.getUserList() this.editDialogVisible = false this.\$message.success('更新司机信息成功') }) }, // 根据 Id 删除对应的司机信息 async removeUserById (userId) { // ... 判断和提示 ... } </pre>
界面展示	

6.6.2 车辆管理模块

模块名称	车辆管理
------	------

功能描述	管理员可以管理系统中的车辆信息
主要算法实现	<p>在前端中通过调用后端接口 findCarCityAndCount 获取车辆城市和数量信息，并将结果存储在 carCityAndCount 中；通过调用后端接口 queryCarList 获取车辆列表信息，根据分页信息、城市、车牌、车型等条件进行查询。将查询结果存储在 carList 中，并更新 total 等分页信息；在后端通过 findCarCityAndCount, queryCarList 两个接口接收前端发出的请求并返回相应的数据</p> <pre> export const findCarCityAndCount = () => { return request({ url: '/Car/findCarCityAndCount', method: 'get' }) } export const queryCarList = (current, size, city, plate, style) => { return request({ url: '/Car/queryCarList', method: 'get', params: { current, size, carTude: city, carPlates: plate, carStyle: style } }) } </pre>
界面展示	

6.7 数据通信子系统的模块设计

6.7.1 全双工通信

模块名称	全双工通信
功能描述	实现前后端的实时通信，传递疲劳检测结果和警告信息
主要算法实现	<p>在前端部分，页面利用 navigator.mediaDevices.getUserMedia 获取用户的摄像头视频流，视频帧通过 canvas 处理后，以 Base64 编码的形式通过使用 Socket.IO 客户端发送到服务器。后端部分使用 Socket.IO 处理来自前端的视频帧数据。</p> <pre> if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) { navigator.mediaDevices.getUserMedia({ video: true }).then((stream) => { this.videoElement.srcObject = stream; this.videoElement.onloadedmetadata = () => { // ... 当视频元数据加载完成时获取尺寸 ... setInterval(() => { context.drawImage(this.videoElement, 0, 0, canvas.width, canvas.height); const dataURL = canvas.toDataURL('image/jpeg'); this.socket.emit('frame', { image: dataURL.split(',')[1], userId: this.userId }); }, 200); }); }); } this.socket.on('response', (data) => { this.imageSrc = 'data:image/jpeg;base64,' + data.data; // ... 更新表格数据 ... // ... 检查眯眼或打哈欠计数是否递增 ... }); </pre>

6.7.2 负载均衡器

模块名称	负载均衡器
功能描述	对入站的请求流量和视频传输流量进行负载均衡
主要算法实现	<p>使用 NGINX 进行反向代理, websocket_backend 上游定义了几个后端服务器, 这些服务器将共同处理通过 /ws 端点进入的 WebSocket 连接</p> <pre> upstream websocket_backend { server cn2.gb.vayki.com:8000; server de.vayki.com:8000; server 4837.us.vayki.com:8000; } server { server_name driver.chenhaotian.top; location /ws { proxy_pass http://websocket_backend; proxy_set_header Upgrade \$http_upgrade; proxy_set_header Connection "upgrade"; proxy_http_version 1.1; proxy_set_header Host \$host; proxy_set_header X-Real-IP \$remote_addr; proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for; proxy_set_header X-Forwarded-Proto \$scheme; proxy_read_timeout 86400; } location / { proxy_pass http://websocket_backend; proxy_set_header Host \$host; proxy_read_timeout 86400; } # ... TLS 证书配置 ... } </pre>

6.7.3 国际网络优化

模块名称	国际网络优化
功能描述	采用三大运营商的优化线路确保全球访问的可靠性
功能实现	由于视频传输的大带宽要求以及国内商业宽带成本现状, 接入具有大陆优化线路的海外服务器负责视频传输, 其广州电信回程路

演示	由如下所示 广州电信 58.60.188.222 22.18 ms * RFC1918 12.18 ms AS4809 [CTE-NET] 德国 黑森州 美因河畔法兰克福 chinatelecom.com.cn 电信 12.57 ms AS23764 德国 黑森州 美因河畔法兰克福 chinatelecomglobal.com 电信 147.34 ms AS23764 德国 黑森州 美因河畔法兰克福 chinatelecomglobal.com 电信 180.81 ms * [CN2-BackBone] 中国 北京市 chinatelecom.cn 电信 195.78 ms * [CN2-BackBone] 中国 广东省 广州市 chinatelecom.cn 电信 195.73 ms AS4134 中国 广东省 佛山市 chinatelecom.com.cn 电信 194.92 ms AS4134 中国 广东省 深圳市 福田区 chinatelecom.com.cn 电信
----	---

7 测试设计及测试报告

7.1 测试设计概述

测试设计环节是确保疲劳驾驶监测系统满足所有规定要求的关键步骤。我们的测试策略旨在评估软件的各个方面，包括功能正确性、性能稳定性、用户界面的友好性以及安全性。测试过程还将重点关注系统的可扩展性和兼容性，确保它能够在不同的硬件和网络环境下稳定运行。

7.2 测试环境设置

测试环境的设置是实施有效测试的基础。我们将搭建与生产环境相似的测试环境，以确保测试结果的准确性和可靠性。这包括用户环境和服务器环境的配置。

类别	组件	描述
用户环境	操作系统	Mac OS 14.2 23C5055b
	SOC	Apple M1 Pro
	浏览器	Microsoft Edge 120.0.2210.77
	摄像头	1080p FaceTime

服务器环境	操作系统	Debian GNU/Linux 12
	CPU	Intel Core Processor (Broadwell)
	CPU 核心数	3
	CPU 频率	2095.320 MHz
	内存	4.28 GiB
	Web 服务器	NGINX 1.22.1
	编程语言	Node.js 16.20.2, Python 3.11

表 7-1 测试环境

7.3 测试用例设计

7.3.1 功能测试

1. 登录验证

目标：验证不同类型用户（司机、管理员）的登录过程是否正常。

步骤：尝试使用有效和无效的凭证登录，包括空密码、错误密码等情况。

预期结果：系统应正确识别并响应各种登录尝试，只有有效凭证才能成功登录。

2. 疲劳监测

目标：确保系统能准确识别疲劳驾驶行为。

步骤：使用预设的疲劳和非疲劳驾驶情景测试，包括闭眼、打哈欠等行为。

预期结果：系统能准确区分疲劳和正常驾驶状态，并在检测到疲劳行为时发出警告。

3. 实时视频监控

目标：检查视频流的稳定性和清晰度。

步骤：在不同网络环境下观察视频流的质量，包括低带宽和高带宽条件。

预期结果：视频流应保持稳定，图像清晰，无明显延迟或断断续续的情况。

7.3.2 性能测试

1. 响应时间

目标：测量系统对不同操作的响应时间。

步骤：执行各种操作，如登录、加载数据、提交请求等，并记录响应时间。

预期结果：所有操作的响应时间应在可接受的范围内，无过长的延迟。

2. 并发处理

目标：验证系统在多用户同时访问时的表现。

步骤：模拟多个用户同时登录、使用疲劳监测和视频监控等功能。

预期结果：系统应能稳定运行，无崩溃或性能显著下降。

7.3.3 安全测试

1. 数据加密

目标：确保敏感信息如用户密码的安全存储和传输。

步骤：检查敏感信息的存储和传输过程是否采用加密。

预期结果：所有敏感数据都应加密处理，防止数据泄露。

7.3.4 用户体验测试

1. 界面友好性

目标：评估用户界面的可用性和易用性。

步骤：测试用户界面的各个方面，包括导航、布局、颜色方案和字体可读性。

预期结果：界面应直观、清晰，用户能够轻松地找到所需功能和信息。

7.4 测试结果与报告

7.4.1 测试结果

模块	项目	测试结果	结果演示	日志信息
功能测试	登录验证	不同类型用户的登录过程均正常		<div>请求 URL: <code>https://192.168.1.100:8080/login</code></div> <div>请求方法: <code>POST</code></div> <div>请求头: <code>Content-Type: application/json</code></div> <div>响应头: <code>Content-Type: application/json</code></div> <div>响应状态: <code>200 OK</code></div> <div>响应内容: <code>{\"success\": true, \"message\": \"Login successful\"}</code></div>
	疲劳监测	正确识别用户的疲劳行为并计数		<div>请求 URL: <code>https://192.168.1.100:8080/fatigue</code></div> <div>请求方法: <code>POST</code></div> <div>请求头: <code>Content-Type: application/json</code></div> <div>响应头: <code>Content-Type: application/json</code></div> <div>响应状态: <code>200 OK</code></div> <div>响应内容: <code>{\"fatigue_level\": 0.8, \"message\": \"Fatigue detected\"}</code></div>
	实时视频监控	前后端数据传输无差错		<div>请求 URL: <code>https://192.168.1.100:8080/video</code></div> <div>请求方法: <code>POST</code></div> <div>请求头: <code>Content-Type: application/json</code></div> <div>响应头: <code>Content-Type: application/json</code></div> <div>响应状态: <code>200 OK</code></div> <div>响应内容: <code>{\"video_url\": \"https://192.168.1.100:8080/video/1\", \"message\": \"Video feed started\"}</code></div>

性能测试	响应时间	网站页面平均加载时间约为 5 秒		
	并发处理	网站在 128 线程的并发请求下运行正常		
安全测试	数据加密	网站所有连接均通过 TLS 加密		
用户体验测试	界面友好性	对未注册用户具有引导提示		

表 7-2 测试结果

7.4.2 测试报告

在对疲劳驾驶监测系统进行全面测试后，我们得到了一系列关键性能指标和用户体验方面的结果。这些测试覆盖了功能性、性能、安全性和用户体验等多个方面，确保系统在各方面达到预期标准。但是，测试过程中也发现了一些需要改进的缺陷。

在功能测试中，系统的核心功能，如登录验证和疲劳行为的准确监测，表现良好。登录过程针对不同用户类型（司机和管理员）都能顺利完成，有效和无效的登录尝试都被正确处理。然而，在疲劳监测方面，尽管系统能准确识别疲劳行为，但疲劳记录的更新存在延迟，影响了系统的及时反应能力。

性能测试方面，系统的响应时间和并发处理能力总体符合要求。但是，网页加载速度慢的问题引起了我们的关注。在测试中发现，平均页面加载时间超过了预期的 2 秒，这可能会影响用户的体验和系统的实时监控效率。

在安全性方面，系统显示出良好的安全防护能力，所有敏感数据传输都通过 TLS 加密，确保了数据的安全性。用户体验测试中，界面设计直观、布局合理，

但用户对网页加载速度慢表达了一定的不满。

虽然疲劳驾驶监测系统在多数测试方面表现出色，但疲劳记录更新不及时和网页加载速度慢的问题仍需改进。我们计划对这些问题进行深入分析，并在后续版本中进行优化。这些测试为我们提供了宝贵的反馈，将指导我们在未来的迭代中改进系统性能和用户体验。

8 项目日常沟通机制及记录

8.1 日常沟通机制

我们使用 QQ 群实时地进行信息交流和讨论。这个 QQ 群成为了日常沟通的主要平台，允许团队成员轻松分享信息和协调工作。在处理紧急或私人事项的情况下，使用 QQ 私聊功能。这种一对一的沟通方式更适合处理敏感或需要快速响应的事务。

为了保持团队同步并及时解决问题，我们设立了每日站会制度。团队成员会在固定的时间通过 QQ 群进行简短的会议，分享各自的工作进展、遇到的难题以及需要协调的事项。这样的日常碰头会帮助团队保持进度和方向的一致性。针对紧急情况或需要即时协调的问题，我们强调随时通过 QQ 私聊或群聊进行沟通。这确保了在关键时刻，团队成员能够快速响应并采取必要的行动，从而有效地处理突发情况。

8.2 沟通内容记录

时间	主题/功能	内容概要	参与人员	负责人	相关记录
12 月 11 日 15 时	选题	确认选题为疲劳驾驶检测	全体成员	陈昊天	
12 月 11 日 16 时	前端框架	确认前端框架使用 Vue	全体成员	戴建雨	
12 月 12 日 11 时	后端框架	确认后端框架使用 Flask，确认项目运行环境	全体成员	陈昊天	

12月13日15时	需求分析	撰写需求分析文档	全体成员	周佳羽	
12月13日16时	后端框架	后端运行参数	戴建雨、陈昊天	陈昊天	
12月13日17时	需求获取	创建框架用例、分工细化	全体成员	戴建雨	
12月13日19时	用户管理	确认账号类型分为司机和交管中心	周佳羽、戴建雨	周佳羽	
12月13日19时	需求分析	分工细化	严贺阳、石查分	严贺阳	
12月13日21时	疲劳预警	驾驶员危险行为声音预警	石查分、陈昊天	陈昊天	
12月14日5时	管理员警示	警示界面 UI 设计	周佳羽、戴建雨	戴建雨	
12月16日11时	代码合并	合并源代码、功能验证和确认	戴建雨、周佳羽、陈昊天	陈昊天	
12月16日13时	人脸识别	人脸识别、疲劳监测的算法端	全体成员	陈昊天	
12月16日13时	项目部署	服务器部署情况研究	戴建雨、陈昊天	陈昊天	
12月16日13时	文档编写	文档编写的模块细化	全体成员	严贺阳	
12月16日14时	前后端交互	确认数据传输方式	全体成员	陈昊天	
12月16日18时	文档编写	总结文档编写成果	周佳羽、严贺阳、陈昊天	严贺阳	

12月17日23时	代码合并	合并源代码、功能验证和确认	戴建雨、周佳羽、陈昊天	陈昊天	
12月18日9时	功能测试	用户登录、疲劳检测功能测试	戴建雨、陈昊天	戴建雨	
12月18日17时	功能测试	前后端 API 联调	周佳羽、戴建雨	周佳羽	
12月18日20时	功能测试	管理员视频监控、服务区距离测试	戴建雨、周佳羽、陈昊天	陈昊天	
12月18日22时	项目部署	服务器生产环境配置	戴建雨、周佳羽、陈昊天	陈昊天	
12月19日11时	文档编写	总结文档编写成果	严贺阳、石查分	严贺阳	
12月19日18时	项目部署	发布项目部署成果	戴建雨、周佳羽、陈昊天	陈昊天	
12月19日19时	功能测试	车辆位置、分页功能测试	戴建雨、周佳羽、陈昊天	周佳羽	
12月19日20时	文档编写	明确模块设计展示规范	严贺阳、石查分、陈昊天	石查分	
12月20日17时	文档编写	总结文档编写成果	严贺阳、石查分	石查分	

			分		
--	--	--	---	--	--

表 8-1 沟通内容记录表

9 项目管理机制及执行记录

9.1 项目管理机制

项目使用远程服务器确保了代码的集中管理和持续集成。通过 SSH，团队成员能够安全地访问和操作远程服务器，这不仅增强了我们的工作效率，也为项目的安全性添加了一层保障。SSH 的使用使得团队成员能够从任何地点进行代码提交、数据同步以及执行必要的命令。

Rsync 工具在我们的项目管理中也发挥了重要作用。它不仅提供了高效的文件同步功能，而且在网络传输中极大地节约了时间和资源。通过 Rsync，我们可以快速地将本地更改同步到远程服务器，或者反向操作，确保了代码库的一致性和最新状态。这种同步机制极大地减少了因文件冲突或版本不一致所导致的问题。

9.2 项目执行记录

时间	内容摘要	提交者	涉及的部分文件列表
12 月 11 日	工程初始化	陈昊天	package-lock.json package.json vue.config.js public/* src/*
12 月 11 日	报警器提示设计文档	石查分	document/*
12 月 11 日	音乐播放器设计文档	石查分	document/*
12 月 13 日	Flask 路由部分、人脸识别算法	陈昊天	src/backend/app.py src/backend/analyze.py
12 月 14 日	管理员 UI 设计、接口实现	周佳羽	src/router/index.js src/components/AdminMenuTree.vue src/views/Admin/* src/views/Login.vue

12 月 14 日	司机 UI 设计、接口实现	戴建雨	src/router/index.js src/components/UserMenuTree.vue src/views/User/* src/views/Register.vue
12 月 15 日	数据库设计文档	陈昊天	document/*
12 月 15 日	Flask SQLite 设计	陈昊天	src/backend/app.py src/backend/data.db src/backend/db.py src/backend/shape_predictor_68_face_landmarks.dat
12 月 16 日	服务区、交规、视频监控文档	戴建雨	document/*
12 月 16 日	需求获取与分析设计文档	严贺阳阳	document/*
12 月 16 日	管理员模块设计文档	周佳羽	document/*
12 月 16 日	服务器生产环境部署配置、疲劳监测前端实现	陈昊天	.env.development .env.production src/utlis/request.js src/views/Admin/Controller/Data.vue src/views/Admin/Controller/VideoSurveillance.vue src/views/User/Controller/Data.vue src/views/User/Controller/VideoSurveillance.vue
12 月 18 日	前后端 API 联调	周佳羽	src/api/admin.js src/api/car.js src/backend/app.py
12 月 18 日	前后端 API 联调	戴建雨	src/api/data.js src/api/driver.js src/backend/app.py
12 月 19 日	系统与模块设计规范	陈昊天	document/*
12 月 19 日	管理员信息查看与提醒	周佳羽	src/views/Admin/Information/Car.vue src/views/Admin/Information/Driver.vue src/views/Admin/Information/Place.vu

			e
12 月 19 日	司机监控数据显示	戴建雨	src/views/User/Information/Introduce. vue src/views/User/Information/Place.vue src/views/User/Information/UserInfo.v ue
12 月 19 日	疲劳驾驶提醒预警、页 面背景	陈昊天	src/views/User/Controller/VideoSurvei llance.vue src/views/User/Controller/VideoSurvei llance.vue src/assets/img/* src/assets/css/* src/backend/analyze.py src/backend/app.py
12 月 20 日	系统设计展示文档	石查分	document/*

表 9-1 项目执行记录

10 团队成员评价及理由

10.1 团队成员评价

在项目过程中，每位团队成员都在自己的岗位上做出了积极的贡献，推动了项目的顺利进行。组长陈昊天对团队成员的评分如下：

姓名	戴建雨	周佳羽	严贺阳阳	石查分
评分	27	26	24	23

表 10-1 团队成员评分表

10.2 成员评价理由

戴建雨，作为 UI 设计师和研发工程师，她负责前端设计与实现，为系统提供了直观、易用的用户界面。她在 UI 设计和前后端功能实现上的工作对提升用户体验起到了关键作用。

周佳羽，作为 UI 设计师和研发工程师，在管理员用户界面的设计和实现上做出了显著的贡献。她的工作确保了系统的高效管理和监控功能。

严贺阳阳，作为系统架构师，参与了系统整体设计，并在测试研发中发挥了重要作用。她的工作对于系统的稳定性和可靠性至关重要。

石查分，作为系统架构师和测试工程师，她在系统架构设计和测试实施上投入了大量工作，为提升系统质量提供了重要保障。

在这个项目中，每个团队成员都发挥了自己独特的技能和才能，共同努力确保了项目的成功，展示了团队合作和个人专业技能的完美结合。

参考文献

- [1] 徐莲, 任小洪, 陈闰雪. 基于眼睛状态识别的疲劳驾驶检测[J]. 科学技术与工程, 2020, 20(20): 8292-8299.
- [2] 牛清宁, 周志强, 金立生, 等. 基于眼动特征的疲劳驾驶检测方法[J]. 哈尔滨工程大学学报, 2015, 36(3): 394-398.
- [3] 朱二华. 基于 Vue.js 的 Web 前端应用研究[J]. 科技与创新, 2017 (20): 119-121.
- [4] 牛作东, 李捍东. 基于 Python 与 flask 工具搭建可高效开发的实用型 MVC 框架[J]. 计算机应用与软件, 2019, 36(7): 21-25.
- [5] 汪添生, 马朝晗, 崔蔚, 等. 基于多线程并发的 JUnit 与 TestNG 测试框架比较[J]. 计算机系统应用, 2017, 26(5): 29-34.