软件设计文档

项目名称：升学指导

2023 年 6 月 15 日

姓名	于海龙
学号	2120220695
完成作业共花费	72小时
ChatGPT 完成任务比例	60%
实现软件功能预计还需要	20天

目录

1 背景介绍

1.1 升学问题现状

当学生决定进入研究生阶段时需要仔细考虑自己的职业目标、兴趣爱好和能力，以及选择最适合自己的学校、专业和导师。这项任务存在一些困难，因为存在太多的选择，并且需要考虑非常多不同的因素，例如学校的声誉、专业的难度、导师的研究领域等等。

当前的学生面临许多的升学问题，这些问题包括了学校选择、专业选择、导师选择等几个方面。作为学生，首先需要确定自己的研究方向，研究方向决定了学生未来的职业发展方向和就业前景；同时学生也会因为选择合适的研究方向更好地发挥自己的优势和兴趣，提高研究效率和成果质量。学校选择也是一项重要任务，因为学校的排名和声誉可以影响到研究生的就业前景和职业发展；与此同时学校所在地的交通、生活条件等也需要考虑，这些因素可能会影响到研究生的生活质量和学习效果。

1.2 软件开发目标

设计一款升学指导软件的目的是帮助学生更好地规划自己的未来，解决他们在升学过程中所面临的各种问题和挑战。目前市面上有很多升学指导的软件，如考研帮、留学Go、升学宝和升学通等。这些市面上的软件具有不同的特色，具体如下：

1. 考研帮：考研帮是一款专门为考研学生提供的软件，它提供了各种考研资讯、历年真题、模拟考试等服务。用户可以通过该软件获取最新的考研政策、报考指南、院校信息等，还可以参加各种在线课程和学习小组，与其他考生交流经验和心得。此外，考研帮还提供了一些实用的工具，如错题本、复习计划、考试倒计时等，帮助学生更好地备考。

2. 留学Go：留学Go是一款为留学生提供服务的软件，它提供了各国留学信息、申请流程、签证办理等方面的指导和建议。用户可以通过该软件了解各国的大学排名、专业设置、学费情况等信息，还可以查看各个大学的官方网站和招生简章，了解具体的申请要求和流程。此外，留学Go还提供了一些实用的功能，如语言测试、文书模板、面试技巧等，帮助学生更好地准备留学申请。

3. 升学宝：升学宝是一款综合性的升学指导软件，它提供了多种服务，包括选校咨询、专业选择、申请指导等。用户可以通过该软件了解各个大学的招生政策、录取标准、课程设置等信息，还可以查看各个专业的就业前景和发展趋势。此外，升学宝还提供了一些实用的功能，如职业测评、志愿填报、面试模拟等，帮助学生更好地规划自己的升学路线。

4. 升学通：升学通是一款为高中生和大学生提供升学指导的软件，它提供了多种服务，

包括选科咨询、专业选择、大学排名等。用户可以通过该软件了解各个大学的招生政策、录取标准、课程设置等信息，还可以查看各个专业的就业前景和发展趋势。

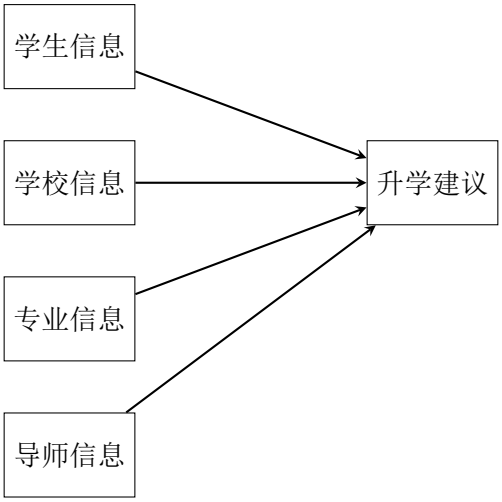

图 1: 获取升学推荐信息

这几款软件存在信息来源不够全面、界面设计不简洁直观和功能不全面等问题。因此我们需要为研究生设计一款简单明了的升学指导软件，如图1所示。这款软件期待的目标包括：简洁明了的界面设计、完备的数据库设计、高效的匹配算法设计和数据安全设计。一款升学指导软件可以帮助学生更好地选择适合自己的研究生阶段的学校、专业和导师，这对于他们的职业发展和人生规划非常重要，因此这款软件具有很高的实用性和价值。

2 项目概述

2.1 系统功能

该软件的主要功能是为用户提供升学建议，并提供学校、专业和导师的信息查询服务。除此之外，还需要提供用户基本信息管理、学校专业信息管理、后台管理和统计分析的功能。为了可以支持后续的学生培养计划，还可以在未来允许职场性格测试、职业发展分析等功能的接入。

2.2 业务描述

2.2.1 用户升学建议查询

用户升学建议查询的主要流程包括：

1. 输入学生信息；包括学习成绩、兴趣爱好、就职意向等，为系统分析数据提供基础支持。
2. 系统分析数据；根据学生输入的信息和数据库中的学校、专业和导师信息，执行匹配算法和推荐算法，对数据进行分析。
3. 生成推荐列表；根据上一步分析的结果，使用Top-K策略选择推荐列表，供用户筛选。
4. 完成选择；用户根据推荐列表完成升学选择。

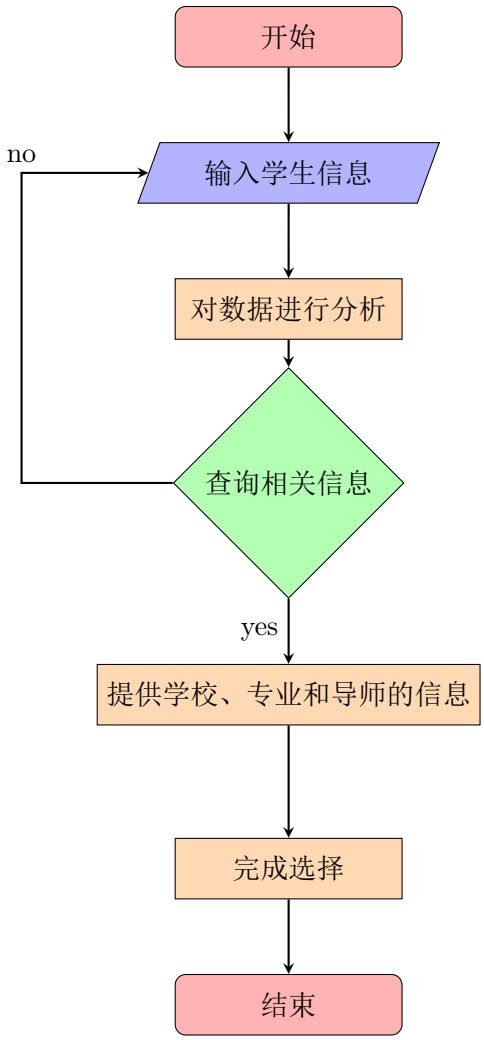

图 2: 输入学生信息查询学校、专业和导师信息流程图

当用户执行一次升学建议查询时，具体的流程如图2所示。

2.2.2 管理员更新学校专业信息

软件管理员更新学校专业和导师信息的主要流程包括：

1. 管理员输入学校信息；用于数据库查询学校信息并执行更新操作。
2. 管理员输入专业信息；用于数据库查询专业信息并执行更新操作。
3. 管理员输入导师信息；用于数据库查询导师信息并执行更新操作。

当管理员用户执行一次学校、专业和导师信息更新时，具体的流程如图3所示。

2.2.3 用户管理

用户注册、登录和登出的主要流程包括：

1. 用户注册；用户需要在注册页面输入用户名、密码、电子邮件地址等信息，并提交注册请求。系统将验证用户输入的信息，如果验证通过，则创建用户账户，并将用户信息保存在用户数据库中。如果验证未通过，则系统将提示用户重新输入信息。
2. 用户登录；用户需要在登录页面输入用户名和密码，并提交登录请求。系统将验证用户输入的信息，如果验证通过，则用户登录系统，可以访问系统提供的服务。如果验证未通过，则系统将提示用户重新输入信息。
3. 用户注销；用户在登录状态下可以选择注销登录，退出系统。用户注销后，系统将清除用户登录信息，确保用户的账户安全。

当用户执行一次账户注册和登录时，具体的流程如图4所示。

2.2.4 统计分析

管理员执行数据统计分析的主要流程包括：

1. 输入登录信息；管理员需要在登录页面输入用户名和密码，并提交登录请求。
2. 验证用户身份；系统将验证管理员输入的信息，如果验证通过，则管理员登录系统，可以访问统计后台数据。如果验证未通过，则系统将提示管理员重新输入信息。
3. 访问统计后台数据；管理员登录后，可以访问统计后台数据，并从数据库中检索相关的统计数据。系统将通过管理员账户和数据库来实现这个过程，以保护管理员的隐私和安全，确保管理员可以方便地查看统计数据。
4. 生成并输出报告；系统将检索相关的统计数据，并生成报告。报告可以显示统计数据的图表、表格和其他相关信息。生成的报告将输出给管理员。管理员可以查看报告，并根据报告中的统计数据做出正确的决策。

当管理员用户执行一次统计数据查看操作时，主要流程如图5所示。

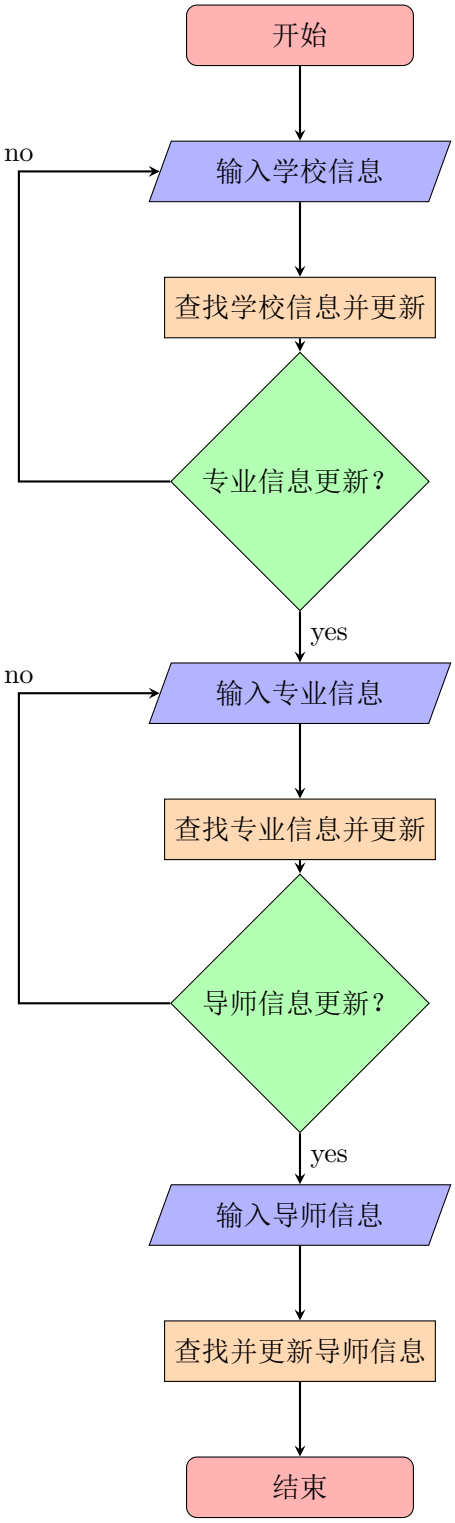

图 3: 更新学校和专业信息流程图

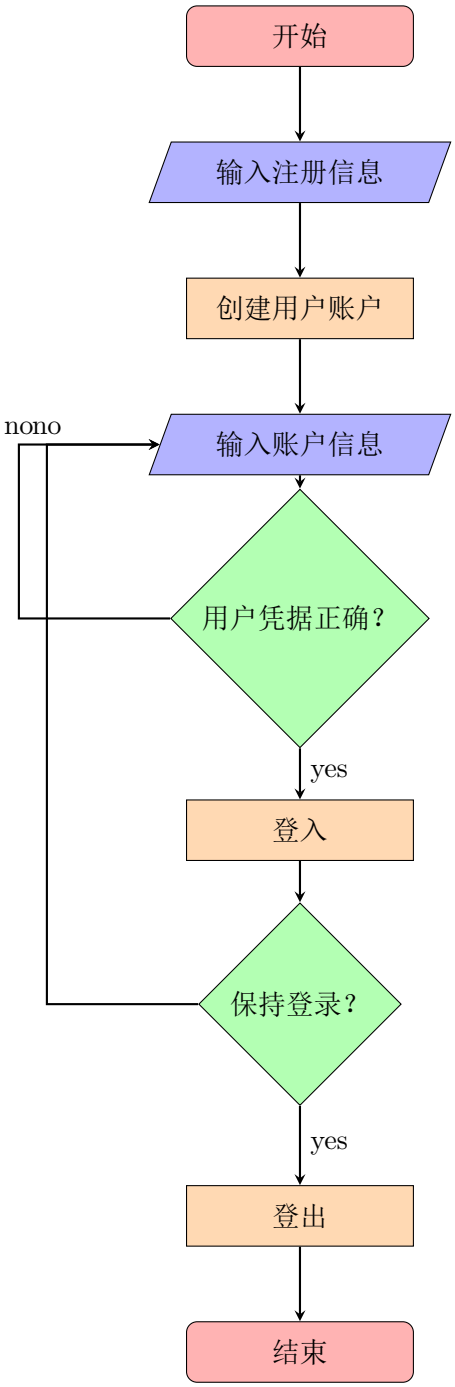

图 4: 用户注册与登录流程图

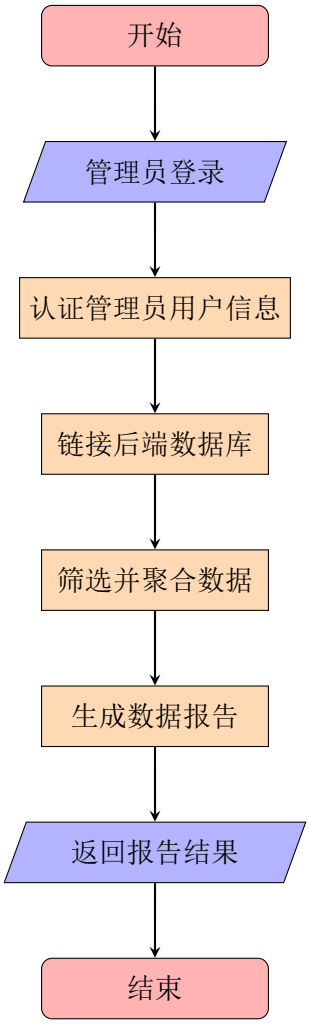

图 5: 管理员查看统计数据

2.3 数据流程描述

2.3.1 数据

在这个软件系统中需要记录保存和管理的数据主要包括：

1. 学生用户信息；学生用户需要提交学生升学相关的数据，包括学生的考试成绩、兴趣爱好、工作目标等内容。除此之外，还需要在数据库中存储用户的凭证信息以供登录使用。

2. 管理员信息；系统数据库需要存储管理员的凭证信息以供登录使用，并且需要存储管理员的操作记录日志，可以在数据出现问题是及时查找问题并恢复。

3. 学校信息；系统数据库需要保存学校信息，以供升学建议的生成。

4. 专业信息；系统数据库需要保存专业信息，专业信息与学校信息关联，以供升学建议的生成。

5. 导师信息；系统数据库需要保存导师信息，导师信息与学校和专业信息关联，以供升学建议的生成。

2.3.2 数据间的关系

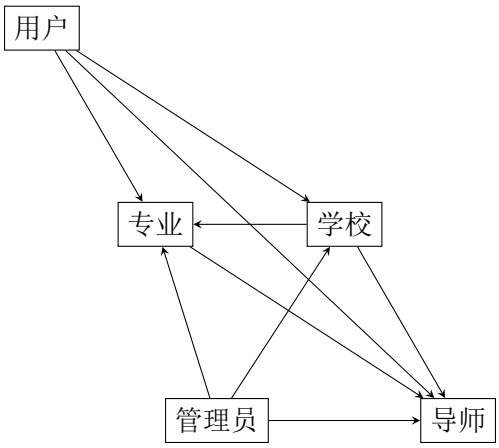

图 6: 存储数据间存在的关系

2.4 用户特点

2.4.1 学生用户

学生用户在升学和职业规划方面的需求可能非常复杂和多样化，例如他们可能需要了解不同学校的录取要求、专业课程设置、就业前景等等。学生用户可能对于数据和统计分析方面的内容比较感兴趣，例如各大学专业的录取率、毕业后就业情况等等，这些数据和统计分析可以帮助他们做出更加明智的决策。

虽然现在的学生都很早就开始接触电脑和互联网，但他们的对软件使用娴熟度仍然各不相同。有些学生可能非常熟练地使用计算机和软件，而另一些学生则可能需要花费更多的时间和精力来适应这些技术。除此之外，学生用户通常喜欢有趣、易于理解的界面和互动方式，因此这个软件系统中的学生用户可能会更喜欢具有可视化效果和游戏化设计的界面和体验。

总体而言，需要给学生用户群体提供简单易懂的可视化界面、清晰全面的推荐信息和稳定可靠的系统平台。

2.4.2 系统管理员

管理员用户通常是教育机构或学校的教育管理人员，具有一定的教育管理和升学规划方面的专业背景和知识。管理员用户需要负责升学指导软件系统的管理和运营，因此需要对这些管理员的操作进行明确完整的日志记录。除此之外，同学生用户一样，他们的对软件使用娴熟度也各不相同。

总体而言，需要给管理员用户群体提供简单易懂的可视化界面、稳定可靠的系统平台和完整的日志记录和恢复系统。

2.5 运行环境要求

2.5.1 前端用户界面

1. Web 网页前端UI，需要基于Blink内核的类Chrome浏览器、基于WebKit的Safari浏览器、基于Gecko内核的FireFox浏览器和IE浏览器。
2. Android 移动应用，需要不低于X版本基于 Linux 内核的 Android 操作系统，或不低于X版本的 HarmonyOS 操作系统。
3. iOS 移动应用，需要不低于X版本的iOS操作系统。
4. 各种小程序应用，与网页前端UI要求相同，需要基于Blink内核的类Chrome浏览器、基于WebKit的Safari浏览器、基于Gecko内核的FireFox浏览器和IE浏览器。

2.5.2 服务器程序

1. 网页服务，需要运行在基于 Linux 内核的操作系统，安装 Apache 和 Nginx 服务完成网站部署、反向代理和负载均衡。
2. 后端服务，需要运行在基于 Linux 内核的操作系统，安装 Java Development Kits 和 Spring Boot 框架。
3. 容器服务，需要运行在基于 Linux 内核的操作系统，安装 docker 支持容器运行、安装 Kubernetes 进行容器管理。
4. 消息队列和缓存，需要运行在基于 Linux 内核的操作系统，安装 Kafka 实现数据流分布式管理、安装 RabbitMQ 实现消息队列处理和事务处理、安装 Redis 实现高速缓存、安装 RocketMQ 实现负载均衡和故障转移。
5. 监控和日志系统，需要监控和日志工具来监控和记录系统的运行状态和性能，以便及时发现和解决问题，常见的监控和日志工具包括Nagios、Zabbix、ELK等。

2.5.3 数据库

1. 数据库管理系统，需要安装数据库管理系统（DBMS）管理数据，包括 MySQL 、PostgreSQL 、Oracle 、SQL Server 等。
2. 数据库连接：需要使用适当的连接方式来连接数据库，常见的连接方式包括 ODB C、JDBC 、ADO.NET等。
3. 编程语言和框架，需要使用编程语言和框架来编写数据库相关的代码，常见的编程语言和框架包括Java和Spring Boot。
4. 数据备份与恢复，需要安装自动备份软件，实现物理备份和逻辑备份，使用完全备份、增量备份、热备份和冷备份结合的方法保障数据库数据的安全。

3 基本设计概述

3.1 用户界面

3.1.1 Web前端界面

这个升学指导软件的前端Web系统需要具备以下要求：

1. 用户友好性：前端Web系统需要具备良好的用户界面和用户体验，易于操作和使用，使用户能够轻松地找到所需信息和功能。

2. 响应式布局：前端Web系统需要具备响应式布局，能够自适应不同的设备和屏幕大小，包括PC端、手机端、平板电脑等，以便用户能够在不同设备上进行访问和使用。
3. 多媒体支持：前端Web系统需要支持多种多媒体格式，包括音频、视频、图片等，以便用户能够获取到更加丰富的信息和资源。
4. 数据可视化：前端Web系统需要支持数据可视化，以便用户能够通过图表、表格等方式更加直观地查看和分析数据和统计信息。
5. 安全性和稳定性：前端Web系统需要具备安全性和稳定性，包括对用户数据和隐私的保护、防止恶意攻击、避免系统崩溃等方面的保障，保证用户的使用体验和数据安全。
6. SEO优化：前端Web系统需要进行搜索引擎优化，包括网站结构、页面内容、关键词等方面的优化，以便更好地被搜索引擎收录和推荐，提高网站的流量和用户数量。
7. 可扩展性和可维护性：前端Web系统需要具备可扩展性和可维护性，以便在未来能够根据需要添加新的功能和模块，同时保证代码的可读性和易于维护。

3.1.2 Android前端界面

1. 用户友好性：Android前端系统需要具备良好的用户界面和用户体验，易于操作和使用，使用户能够轻松地找到所需信息和功能。
2. 响应式布局：Android前端系统需要具备响应式布局，能够自适应不同的设备和屏幕大小，包括手机、平板电脑等，以便用户能够在不同设备上进行访问和使用。
3. 多媒体支持：Android前端系统需要支持多种多媒体格式，包括音频、视频、图片等，以便用户能够获取到更加丰富的信息和资源。
4. 数据可视化：Android前端系统需要支持数据可视化，以便用户能够通过图表、表格等方式更加直观地查看和分析数据和统计信息。
5. 安全性和稳定性：Android前端系统需要具备安全性和稳定性，包括对用户数据和隐私的保护、防止恶意攻击、避免应用程序崩溃等方面的保障，保证用户的使用体验和数据安全。
6. 多语言支持：Android前端系统需要支持多语言，以便用户能够选择自己熟悉的语言进行使用。
7. 可扩展性和可维护性：Android前端系统需要具备可扩展性和可维护性，以便在未来能够根据需要添加新的功能和模块，同时保证代码的可读性和易于维护。

8. 适配不同操作系统版本：Android前端系统需要适配不同的Android系统版本，以便用户能够在不同版本的Android系统上使用应用程序。

3.1.3 iOS前端界面

1. 用户友好性：iOS前端系统需要具备良好的用户界面和用户体验，易于操作和使用，使用户能够轻松地找到所需信息和功能。
2. 响应式布局：iOS前端系统需要具备响应式布局，能够自适应不同的设备和屏幕大小，包括iPhone、iPad等，以便用户能够在不同设备上进行访问和使用。
3. 多媒体支持：iOS前端系统需要支持多种多媒体格式，包括音频、视频、图片等，以便用户能够获取到更加丰富的信息和资源。
4. 数据可视化：iOS前端系统需要支持数据可视化，以便用户能够通过图表、表格等方式更加直观地查看和分析数据和统计信息。
5. 安全性和稳定性：iOS前端系统需要具备安全性和稳定性，包括对用户数据和隐私的保护、防止恶意攻击、避免应用程序崩溃等方面的保障，保证用户的使用体验和数据安全。
6. 多语言支持：iOS前端系统需要支持多语言，以便用户能够选择自己熟悉的语言进行使用。
7. 可扩展性和可维护性：iOS前端系统需要具备可扩展性和可维护性，以便在未来能够根据需要添加新的功能和模块，同时保证代码的可读性和易于维护。
8. 适配不同iOS系统版本：iOS前端系统需要适配不同的iOS系统版本，以便用户能够在不同版本的iOS系统上使用应用程序。

3.1.4 小程序前端界面

1. 用户友好性：小程序前端系统需要具备良好的用户界面和用户体验，易于操作和使用，使用户能够轻松地找到所需信息和功能。
2. 响应式布局：小程序前端系统需要具备响应式布局，能够自适应不同的设备和屏幕大小，包括手机、平板电脑等，以便用户能够在不同设备上进行访问和使用。

3. 多媒体支持：小程序前端系统需要支持多种多媒体格式，包括音频、视频、图片等，以便用户能够获取到更加丰富的信息和资源。

4. 数据可视化：小程序前端系统需要支持数据可视化，以便用户能够通过图表、表格等方式更加直观地查看和分析数据和统计信息。

5. 安全性和稳定性：小程序前端系统需要具备安全性和稳定性，包括对用户数据和隐私的保护、防止恶意攻击、避免应用程序崩溃等方面的保障，保证用户的使用体验和数据安全。

6. 多语言支持：小程序前端系统需要支持多语言，以便用户能够选择自己熟悉的语言进行使用。

7. 可扩展性和可维护性：小程序前端系统需要具备可扩展性和可维护性，以便在未来能够根据需要添加新的功能和模块，同时保证代码的可读性和易于维护。

8. 适配不同小程序平台：小程序前端系统需要适配不同的小程序平台，例如微信小程序、支付宝小程序等，以便用户能够在不同的小程序平台上使用应用程序。

3.2 后端程序设计

3.2.1 网页服务

1. 可靠性和稳定性：Web服务器需要具备高可靠性和稳定性，以保证系统的稳定运行和数据的安全。

2. 高并发处理能力：Web服务器需要具备高并发处理能力，能够同时处理大量用户请求和访问，以保证系统的响应速度和性能。

3. 安全性和防护能力：Web服务器需要具备安全性和防护能力，包括防止恶意攻击、保护用户数据和隐私、防止系统被攻击等方面的保障。

4. 负载均衡：Web服务器需要支持负载均衡，以便将用户请求和访问均匀地分配到不同的服务器节点上，提高系统的并发能力和性能。

5. 高可用性：Web服务器需要具备高可用性，包括故障自动切换、容错机制、备份和恢复等方面的保障，以保证系统的连续性和可用性。

6. CDN加速：Web服务器需要支持CDN加速，以便能够通过CDN分发内容和资源，提高系统的访问速度和性能。

7. 数据备份和恢复：Web服务器需要支持数据备份和恢复，以保证数据的安全和完整性。

8. 高效的日志管理：Web服务器需要具备高效的日志管理，能够对系统运行日志进行收集、存储、分析和展示，以便进行系统管理和优化。
9. 高效的缓存管理：Web服务器需要具备高效的缓存管理，能够对系统的缓存进行管理和优化，提高系统的性能和并发能力。

3.2.2 后端服务

1. 可扩展性和可维护性：后端服务需要具备可扩展性和可维护性，以便在未来能够根据需要添加新的功能和模块，同时保证代码的可读性和易于维护。
2. 安全性和稳定性性：后端服务需要具备安全性和稳定性，包括对用户数据和隐私的保护、防止恶意攻击、避免应用程序崩溃等方面的保障，保证用户的使用体验和数据安全。
3. 高并发处理能力：后端服务需要具备高并发处理能力，能够同时处理大量用户请求和访问，以保证系统的响应速度和性能。
4. 缓存管理：后端服务需要具备缓存管理，能够对数据和资源进行缓存处理，提高系统的响应速度和性能。
5. 数据库管理：后端服务需要对数据库进行管理，包括数据的存储、查询、更新和删除等操作，以便实现数据的持久化和管理。
6. RESTful API设计：后端服务需要设计和实现RESTful API接口，以便前端和其他系统能够通过API接口与后端服务进行通信和数据交互。
7. 消息队列：后端服务需要支持消息队列，以便能够实现异步处理、任务调度和事件驱动等功能。
8. 日志记录和监控：后端服务需要支持消息队列，以便能够实现异步处理、任务调度和事件驱动等功能。
9. 安全认证和授权：后端服务需要进行安全认证和授权，以保证系统的安全性和用户数据的安全。

3.2.3 数据库服务

1. 可靠性和稳定性：数据库服务需要具备高可靠性和稳定性，以保证数据的安全和可靠性。
2. 高性能和高可用性：数据库服务需要具备高性能和高可用性，能够支持大量并发访问和高效的数据处理，同时保证系统的连续性和可用性。
3. 数据库备份和恢复：数据库服务需要支持数据备份和恢复，以保证数据的安全和完整性。
4. 数据库性能优化：数据库服务需要进行性能优化，包括索引优化、查询优化、缓存优化等方面的优化，以提高数据库的响应速度和性能。
5. 数据库安全管理：数据库服务需要进行安全管理，包括用户认证和授权、数据加密、访问控制等方面的管理，以保证数据的安全和隐私。
6. 数据库监控和管理：数据库服务需要进行监控和管理，包括数据库状态监控、性能分析、容量规划等方面的管理，以保证数据库的稳定性和可用性。
7. 数据库升级和迁移：数据库服务需要支持数据库升级和迁移，以便能够根据需要进行系统升级和数据迁移。
8. 数据库备份和恢复策略：数据库服务需要制定数据库备份和恢复策略，包括备份周期、备份方式、备份存储和恢复流程等方面的策略，以保证数据的安全和可靠性。
9. 数据库容灾和故障恢复：数据库服务需要具备容灾和故障恢复能力，包括故障自动切换、容错机制、备份和恢复等方面的保障，以保证数据的连续性和可用性。

3.3 功能需求

3.3.1 升学指导

1. 业务定义与描述：该升学软件是一款提供升学辅助服务的应用程序，旨在帮助用户了解各种高等教育课程和学位的信息，包括申请条件、申请过程和入学要求。该应用程序将提供以下功能：学校搜索、课程搜索、申请流程指南、入学要求和申请条件。
2. 适用的用户类型，指操作本功能所需的授权：该应用程序面向认证的学生用户，通过学生授权即可使用。
3. 功能项的主要页面或样式：该应用程序将提供一个网页界面，其中包括学校搜索、课程搜索、申请流程指南、入学要求和申请条件等功能项，用户可以在该网页上进行操作。

4. 约束条件和特殊考虑：该应用程序不涉及需要计算机化的功能，所有信息都是基于已有的教育机构和课程信息提供的。由于教育机构和课程信息可能会发生变化，因此系统需要定期更新数据以保持准确性。

输入：学校名称、地理位置、学校类型等搜索条件；学生成绩和学生信息，用户的查找优先级设置。

输出：学校搜索结果，包括学校名称、地理位置、学校类型、排名等信息；申请流程指南，包括申请时间、材料提交、面试安排等信息；入学要求，包括学历要求、语言要求、考试要求等信息；申请条件，包括申请资格、申请材料、申请费用等信息。

在异常情况下，例如用户输入的搜索条件不符合要求或者搜索结果为空时，系统将会提示用户重新输入或者提供其他相关信息。

3.3.2 后台管理

1. 业务定义与描述：该升学软件后台管理系统是一款提供升学辅助服务的应用程序，旨在帮助管理员管理学校、专业、导师和申请流程信息。该应用程序将提供以下功能：学校管理、专业管理、导师管理、申请流程管理、用户管理和数据统计。

2. 该应用程序面向管理员用户开放，需要授权才能使用。

3. 功能项的主要页面或样式：该应用程序将提供一个网页界面，其中包括学校管理、专业管理、导师管理、申请流程管理、用户管理和数据统计等功能项，管理员可以在该网页上进行操作。

4. 约束条件和特殊考虑：该应用程序需要保护用户数据的安全性和隐私，因此需要采用安全措施来防止未经授权的访问。另外，由于学校、专业、导师和申请流程信息可能会发生变化，因此需要及时更新数据库信息，以确保数据的准确性和时效性。同时，需要遵守数据保护和隐私法规，保护用户隐私信息。另外，应用程序需要具有良好的可扩展性和可维护性，以适应未来的需求变化和技术升级。

输入：学校信息、专业信息、导师信息、申请流程信息、用户信息。

输出：学校信息、专业信息、导师信息、申请流程信息、用户信息、统计信息。

在异常情况下，例如用户输入的数据格式不正确或者保存失败时，系统将会提示管理员重新输入或者提供其他相关信息。

3.4 非功能需求

3.4.1 系统可用性

1. 系统可靠性：系统需要具备高可靠性和稳定性，以确保系统的持续运行和提供可靠的服务。
2. 用户友好性：系统需要采用用户友好的界面设计和操作流程，以提高用户的使用体验。
3. 可访问性：系统需要具备可访问性，以支持不同类型的用户（如残障人士）使用系统。
4. 系统可维护性：系统需要具备可维护性，以方便后续的维护和升级工作。
5. 系统安全性：系统需要具备高安全性，以保护用户的数据和隐私信息。
6. 系统可扩展性：系统需要具备可扩展性，以应对未来的业务需求变化和技术升级。

3.4.2 系统性能

1. 响应时间：系统响应用户请求的时间需要尽量短，以提高用户体验。
2. 吞吐量：系统需要具备处理大量数据的能力，以确保系统的高效性。
3. 并发处理能力：系统需要具备同时处理多个用户请求的能力，以支持多用户同时访问系统。
4. 资源利用率：系统需要合理利用资源，如CPU、内存、磁盘等，以确保系统的高效性和稳定性。
5. 性能测试：系统需要进行全面的性能测试，包括负载测试、压力测试、性能监控等，以评估系统的性能和稳定性。

3.4.3 安全性

该系统需要确保用户数据的安全性和隐私，防止未经授权的访问和数据泄露等问题。系统应采用安全措施，如加密技术、身份验证、访问控制和审计等，以保护用户数据的安全性和隐私。

1. 用户身份验证：为了保证系统的安全性，需要对用户进行身份验证，以确保只有授权用户才能访问系统。可以采用用户名和密码、短信验证码、指纹识别等多种身份验证方式，以增强系统的安全性。

2. 数据加密：对于敏感数据和隐私信息，需要采用数据加密技术进行保护。可以采用对称加密算法或非对称加密算法对数据进行加密，以确保数据在传输和存储过程中的安全性。

3. 访问控制：为了保护系统的安全性，需要对用户进行访问控制，以确保只有授权用户才能访问系统的特定功能或数据。可以采用基于角色或基于权限的访问控制机制，以确保系统的安全性。

4. 审计日志：为了跟踪系统的操作和异常行为，需要记录系统的审计日志。可以记录用户的登录和退出、操作行为、异常事件等信息，以便对系统进行监控和审计。

5. 异常处理：对于系统的异常情况，需要采取相应的处理措施，以保证系统的安全性。例如，系统出现SQL注入、跨站脚本攻击等安全漏洞时，需要采取相应的修复措施，防止黑客入侵和数据泄露等问题。

6. 数据备份：为了保证数据的安全性和可靠性，需要对数据进行备份。可以采用定期备份、增量备份等方式，以确保数据在系统故障或数据丢失的情况下能够及时恢复。

3.4.4 可靠性

该系统需要保证高可靠性，确保系统运行的稳定性和可用性。系统应具有故障恢复和备份机制，以确保在系统故障或数据丢失的情况下，能够及时恢复系统运行和数据。

1. 故障恢复：为了保证系统的可靠性，系统需要具备故障恢复的能力。当系统出现故障时，应能够及时发现并采取相应的措施进行恢复，以保证系统可以正常运行。例如，可以采用备份机制、冗余设计等方式增强系统的故障恢复能力。

2. 数据备份和恢复：为了保证数据的可靠性和完整性，需要对数据进行备份和恢复。可以采用定期备份、增量备份等方式，以确保数据在系统故障或数据丢失的情况下能够及时恢复。同时，需要测试数据恢复的可行性和正确性，以确保数据能够完全恢复。

3. 系统监控和报警：为了及时发现系统的异常情况，需要对系统进行监控和报警。可以采用实时监控、日志审计等方式，及时发现系统的异常情况并进行处理。同时，需要设置报警机制，当系统出现异常时，能够及时通知相关人员进行处理，以保证系统的可靠性和稳定性。

4. 代码质量和测试：为了保证系统的可靠性，需要具备高质量的代码和测试。可以采用代码规范、代码审查等方式，确保代码的可读性、可维护性和可靠性。同时，需要进行全面的测试，包括单元测试、集成测试、系统测试等，以确保系统的功能和性能均符合要求。

5. 负载均衡：为了保证系统的可靠性和性能，需要采用负载均衡技术，以均衡系统的负载和流量。可以采用硬件负载均衡器或软件负载均衡器，以确保系统的高可用性和稳定性。

6. 高可用性设计：为了保证系统的可靠性和稳定性，需要采用高可用性设计。可以采用冗余设计、灾备设计等方式，以确保系统在出现故障时能够自动切换到备份系统，保证系统的可用性和稳定性。

7. 容错性和可恢复性：为了保证系统的可靠性和容错性，需要采用容错和可恢复性设计。可以采用异常处理机制、事务处理机制、数据校验等方式，以确保系统在出现异常情况时能够自动恢复并保证数据的完整性和正确性。

3.4.5 可维护性

该系统需要具有良好的可维护性，以保证系统的可持续发展。系统应具有清晰的代码结构、易于维护和扩展的架构设计和完善的文档和注释，以方便后续开发和维护工作。

1. 可读性和可维护性：为了保证系统的可维护性，需要具备高质量的代码和易于维护的结构。可以采用代码规范、代码注释等方式，提高代码的可读性和可维护性。同时，需要对代码进行优化和重构，以确保代码的可扩展性和可维护性。

2. 设计模式和架构：为了保证系统的可维护性，需要采用合适的设计模式和架构，以提高系统的可扩展性和可维护性。可以采用MVC、MVP、MVVM等设计模式，以及分层、微服务、SOA等架构模式，以确保系统的可维护性和可扩展性。

3. 测试和调试：为了保证系统的可维护性，需要进行全面的测试和调试工作。可以采用自动化测试、单元测试、集成测试、系统测试等方式，以及调试工具和技术，以确保系统的稳定性和可维护性。

4. 文档和知识库：为了保证系统的可维护性，需要建立完善的文档和知识库，以便后续的维护和升级工作。可以建立系统的设计文档、用户手册、API文档等，以及建立知识库和FAQ，以提供后续维护人员使用。

5. 版本控制和发布管理：为了保证系统的可维护性和升级能力，需要采用合适的版本控制和发布管理机制。可以采用Git、SVN等版本控制工具，以及自动化发布工具和流程，以确保系统的可维护性和升级能力。

6. 代码审查和重构：为了保证系统的可维护性，需要进行代码审查和重构工作。可以采用代码审查工具和流程，以及重构工具和技术，对系统的代码进行优化和重构，以提高代码的可读性、可维护性和可扩展性。

7. 合理的技术选型和架构设计：为了保证系统的可维护性，需要采用合理的技术选型和架构设计。可以采用成熟、稳定、易于维护的技术和工具，以及合理的架构设计，以确保系统的可维护性和稳定性。

3.4.6 可扩展性

该系统需要具有良好的可扩展性，以应对未来的业务需求变化和技术升级。系统应具有模块化的架构设计、可插拔的功能组件和灵活的配置选项，以方便系统的扩展和升级。

1. 设计良好的架构：为了保证系统的可扩展性，需要采用设计良好的架构，以支持系统的可扩展性。可以采用分层架构、微服务架构、SOA架构等，以确保系统的可扩展性和灵活性。
2. 模块化设计：为了保证系统的可扩展性，需要采用模块化的设计方式，以支持模块的独立开发、测试和部署。可以采用模块化的编程方式，以确保系统的可扩展性和灵活性。
3. 弹性设计：为了保证系统的可扩展性，需要采用弹性设计，以支持系统的自适应和自我修复能力。可以采用负载均衡、自动扩展、容错机制等，以确保系统的可扩展性和稳定性。
4. 高可用性设计：为了保证系统的可扩展性，需要采用高可用性设计，以支持系统的持续运行和可靠性。可以采用冗余设计、灾备设计等，以确保系统的可扩展性和稳定性。
5. 采用云计算和容器技术：为了保证系统的可扩展性，可以采用云计算和容器技术，以支持系统的弹性扩展和部署。可以采用云服务器、容器编排工具等，以确保系统的可扩展性和灵活性。
6. 技术选型和架构设计：为了保证系统的可扩展性，需要采用合适的技术选型和架构设计。可以采用成熟、稳定、易于扩展的技术和工具，以及合理的架构设计，以确保系统的可扩展性和稳定性。
7. 预测和规划：为了保证系统的可扩展性，需要进行预测和规划。可以通过对业务需求和用户需求的分析，预测系统未来的需求变化和业务增长，以进行合理的规划和扩展工作。

3.4.7 兼容性

该系统需要具有良好的兼容性，以适应不同的硬件和软件环境。系统应支持不同的操作系统、浏览器和设备，以确保系统能够在不同的环境下正常运行和展示。

1. 浏览器和操作系统兼容性：为了保证系统的可兼容性，需要考虑不同浏览器和操作系统的兼容性。可以采用标准化的Web技术和CSS样式，以确保系统在不同的浏览器和操作系统上都能正常运行。

2. 移动设备兼容性：为了保证系统的可兼容性，需要考虑不同移动设备的兼容性。可以采用响应式设计或移动优先的设计，以确保系统在不同的移动设备上都能正常运行。

3. 数据库兼容性：为了保证系统的可兼容性，需要考虑不同数据库的兼容性。可以采用标准化的SQL语句和ORM框架，以确保系统在不同的数据库上都能正常运行。

4. 第三方组件和集成兼容性：为了保证系统的可兼容性，需要考虑与第三方组件和服务的兼容性。可以进行充分的测试和集成工作，以确保系统与第三方组件和服务的兼容性。

5. 跨平台和跨语言兼容性：为了保证系统的可兼容性，需要考虑跨平台和跨语言的兼容性。可以采用标准化的API和协议，以确保系统在不同的平台和语言上都能正常运行。

6. 版本兼容性：为了保证系统的可兼容性，需要考虑不同版本之间的兼容性。可以进行版本管理和测试，以确保系统在不同版本之间的兼容性。

7. 安全兼容性：为了保证系统的可兼容性，需要考虑安全方面的兼容性。可以采用标准化的加密算法和安全协议，以确保系统在不同安全环境下都能正常运行。

3.4.8 隐私

隐私是指个人的个人信息、行为和身份等不被他人获取、使用和泄露的权利。在软件开发中，隐私是指系统需要保护个人信息和隐私，以确保用户的隐私权不受侵犯。

1. 数据保护和加密：为了保护用户的隐私，需要对用户的个人信息和敏感数据进行保护和加密。可以采用数据加密技术、SSL协议等方式，以确保用户的个人信息和敏感数据不被他人获取和使用。

2. 访问控制和权限管理：为了保护用户的隐私，需要对用户的个人信息和敏感数据进行访问控制和权限管理。可以采用RBAC、ABAC等访问控制机制，以确保用户的个人信息和敏感数据只被授权的人员访问和使用。

3. 匿名化和脱敏：为了保护用户的隐私，需要对用户的个人信息进行匿名化和脱敏处理。可以采用哈希算法、脱敏工具等方式，以确保用户的个人信息不被他人获取和使用。

4. 合规和法律保护：为了保护用户的隐私，需要确保系统符合相关的隐私法规和标准。可以采用GDPR、CCPA等隐私法规和标准，以确保系统的合规性和用户的隐私权得到法律保护。

5. 安全审计和监控：为了保护用户的隐私，需要对系统进行安全审计和监控，以发现和防止潜在的隐私问题和漏洞。可以采用安全审计工具、防火墙等方式，以确保系统的安全性和用户的隐私权得到保护。

6. 用户知情权和选择权：为了保护用户的隐私，需要尊重用户的知情权和选择权。可以提供明确的隐私政策和用户协议，以告知用户系统如何收集、使用和保护用户的个人信息，以及提供用户选择是否参与的权利。

7. 数据删除和销毁：为了保护用户的隐私，需要对用户的个人信息进行删除和销毁。可以采用数据删除工具和安全销毁机制，以确保用户的个人信息得到有效删除和销毁，避免信息泄露和滥用。

3.5 接口说明

3.5.1 硬件接口

升学指导软件是一个纯软件系统，不需要额外增加硬件接口与外界信息进行交互。

3.5.2 软件接口

这个升学指导软件需要根据学生输入的成绩信息、兴趣、职业发展目标等内容获取升学建议。除此之外，还需要用户注册、验证码获取、用户登录、用户登出和用户注销接口。

而对于管理员而言，需要管理员注册、验证码获取、管理员登录、管理员登出、数据管理和获取统计数据接口。

3.5.3 通信接口

对于这个升学指导软件而言，需要设计数据库通信接口，以便于获取学校信息、专业信息、导师信息、用户信息和录取信息。

4 软件架构

4.1 需求规定

4.1.1 个人信息

学生个人信息需要记录学生的考试成绩、兴趣爱好和职业发展目标，以便生成有效的升学指导意见。

4.1.2 学校信息

包括高校的名称、所在地区、类型、等级、主页网址等信息，以便系统能够了解各高校的基本情况和概况。

4.1.3 专业信息

包括高校开设的专业名称、学制、培养目标、课程设置等信息，以便系统能够了解各专业的基本情况和概况。

4.1.4 导师信息

包括导师的姓名、职称、所属高校、所在科研机构、研究方向、科研成果等信息，以便系统能够了解各导师的基本情况和概况。

4.1.5 用户信息

包括用户的全局唯一识别码、经过单向加密的用户密码、用户基本信息等内容。

4.1.6 录取信息

包括各高校的招生政策和招生计划，例如录取分数线、招生人数、录取比例等信息，以便系统能够为研究生提供准确的升学建议和指导。还有各高校和导师的评价和口碑，例如学术水平、教学质量、师生关系等方面的评价，以便系统能够为研究生提供更全面和准确的升学建议和指导。

4.2 运行环境

4.2.1 用户网站

研究生升学指导软件的用户网站是用户使用软件的主要入口，用户可以在网站上注册、登录、查看升学建议等。用户网站需要运行在稳定的服务器上，具备高可用性、高并发性和安全性等特点。

4.2.2 Android设备

研究生升学指导软件需要在Android设备上运行，以便用户能够在手机或平板电脑上方便地使用软件。因此，软件需要兼容多种操作系统和设备，具有良好的稳定性和性能表现，以确保用户体验。

4.2.3 iOS设备

研究生升学指导软件需要在iOS设备上运行，以便用户能够在iPhone或iPad上方便地使用软件。因此，软件需要兼容多种操作系统和设备，具有良好的稳定性和性能表现，以确保用户体验。

4.2.4 网站后台管理

研究生升学指导软件的网站后台是系统管理人员使用的管理平台，用于管理用户信息、专业信息、学校和导师信息等。网站后台需要具备高效的管理功能、用户权限管理、安全性等特点，以确保系统管理的高效性和安全性。

4.2.5 后台数据库

研究生升学指导软件需要使用数据库服务来存储和管理用户信息、专业信息、学校和导师信息等数据。数据库服务需要具备高可靠性、高可用性、高性能和安全等特点，以确保数据的完整性和安全性。

4.3 软件架构

研究生升学指导软件系统的架构设计采用了典型的三层架构，即表示层、业务逻辑层和数据访问层，结构如图7所示。

1. 表示层：表示层是用户和系统交互的界面，包括网站、安卓设备和iOS设备等。表示层主要负责接收用户请求并将请求发送到业务逻辑层进行处理，同时将业务逻辑层返回的数据进行展示和呈现。

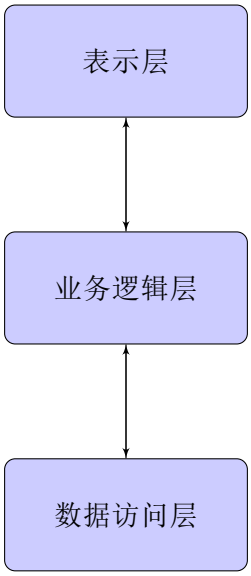

图 7: 软件架构图

2. 业务逻辑层：业务逻辑层是系统的核心层，主要负责处理业务逻辑和数据处理。业务逻辑层包括升学信息管理模块、用户管理模块、学校和专业管理模块、导师管理模块等，负责处理用户提交的请求、管理数据、进行数据分析和处理等操作。

3. 数据访问层：数据访问层负责与数据库进行交互，包括数据的存储、读取和更新等操作。数据访问层采用ORM框架进行数据操作，以提高数据访问效率和便捷性。

4.4 接口设计

对于这样一个庞大的软件系统，我们可以选择使用Spring Boot框架进行后端开发和接口设计工作。

Spring Boot是一个快速开发工具，它可以帮助开发者快速搭建Spring框架，启动一个Web容器，继承了原有Spring框架的核心功能，同时还提供了很多方便的特性，如起步依赖、内嵌容器支持、Actuator监控等。Spring Boot可以快速构建应用程序，减少了开发人员的工作量和时间成本。此外，Spring Boot还提供了丰富的自动配置功能，可以根据不同的需求自动配置应用程序，避免了繁琐的配置工作。另外，Spring Boot还提供了很好的可扩展性和灵活性，可以根据不同的需求进行定制。

4.4.1 用户

使用 @RestController 和 @RequestMapping 注解定义了一个用户认证控制器，包含以下功能：

1. 注册新用户：使用 @PostMapping"/*register*" 注解定义一个 POST 方法，使用 @RequestParam 注解指定了传入的用户名和密码参数名，如果注册成功则返回true，否则返回 false。
2. 用户登录：使用 @PostMapping"/*login*" 注解定义一个 POST 方法，使用 @RequestParam 注解指定了传入的用户名和密码参数名，如果登录成功则返回 true，否则返回 false。
3. 用户登出：使用 @PostMapping"/*logout*" 注解定义一个 POST 方法，使用 @RequestParam 注解指定了传入的用户名参数名，根据实际需求在客户端中删除用户的登录状态即可。
4. 用户注销账户：使用 @DeleteMapping"/*delete*" 注解定义一个 DELETE方法，使用 @RequestParam 注解指定了传入的用户名和密码参数名，如果验证成功则删除该用户的账户信息，并返回 true，否则返回 false。

4.4.2 升学建议

使用 @RestController 和 @RequestMapping 注解定义了一个查询控制器，并实现了以下功能：

1. 获取学生成绩：使用 @GetMapping"/*grades*" 注解定义了一个 GET 方法，使用 @RequestParam 注解指定了传入的学生ID参数名，根据实际需求从数据库中查询学生成绩并返回。
2. 获取学生兴趣：使用 @GetMapping"/*interests*" 注解定义了一个 GET 方法，使用 @RequestParam 注解指定了传入的学生ID参数名，根据实际需求从数据库中查询学生的兴趣并返回。
3. 获取学生职业目标：使用 @GetMapping"/*career*" 注解定义了一个 GET 方法，使用 @RequestParam 注解指定了传入的学生ID参数名，根据实际需求从数据库中查询学生的职业目标并返回。
4. 获取目标学校信息：使用 @GetMapping"/*school*" 注解定义了一个 GET 方法，使用 @RequestParam 注解指定了传入的学校名称参数名，根据实际需求从数据库中查询目标学校信息并返回。

表 1: 管理员权限表

权限代码	授权读取	授权写入
0111	禁止	允许
0110	禁止	禁止
0100	禁止	禁止
0101	禁止	允许

4.4.3 管理员

管理员用户注册、登录和登出的接口与普通用户接口类似。而对于学校、专业和导师信息，需要补全增删改查四种数据库操作接口。管理员权限设计如表1所示。

4.5 数据库设计

4.5.1 用户信息

表 2: 用户信息表

列名	类型	描述
id	int	用户ID，主键
username	varchar50	用户名
password	varchar50	密码
email	varchar50	邮箱
role	varchar20	用户角色

4.5.2 学校信息

表 3: 学校信息表

列名	类型	描述
id	int	学校ID 主键
name	varchar50	学校名称
address	varchar100	学校地址

表 4: 专业信息表

列名	类型	描述
id	int	专业ID 主键
name	varchar50	专业名称
school_id	int	学校ID 外键

4.5.3 专业信息

4.5.4 导师信息

表 5: 导师信息表

列名	类型	描述
id	int	导师ID，主键
name	varchar50	导师名称
title	varchar50	导师职称
major_id	int	专业ID，外键

5 代码

5.1 用户

```
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;

@RestController
@RequestMapping("/auth")
public class UserAuthenticationController {
    private HashMap<String, String> registeredUsers;

    public UserAuthenticationController() {
        this.registeredUsers = new HashMap<String, String>();
    }

    @PostMapping("/register")
    public boolean registerUser(@RequestParam String username, @RequestParam
        String password) {
```

```
        if (this.registeredUsers.containsKey(username)) {
            return false;
        } else {
            this.registeredUsers.put(username, password);
            return true;
        }
    }

    @PostMapping("/login")
    public boolean loginUser(@RequestParam String username, @RequestParam String
        password) {
        if (this.registeredUsers.containsKey(username) &&
            this.registeredUsers.get(username).equals(password)) {
            return true;
        } else {
            return false;
        }
    }

    @PostMapping("/logout")
    public void logoutUser(@RequestParam String username) {
        delete
    }

    @DeleteMapping("/delete")
    public boolean deleteUser(@RequestParam String username, @RequestParam
        String password) {
        if (this.registeredUsers.containsKey(username) &&
            this.registeredUsers.get(username).equals(password)) {
            this.registeredUsers.remove(username);
            return true;
        } else {
            return false;
        }
    }
}
```

5.2 升学建议

```
import org.springframework.web.bind.annotation.*;
```

```java
import java.util.ArrayList;
import java.util.HashMap;

@RestController
@RequestMapping("/search")
public class SchoolSearchController {
    private HashMap<String, School> schools;

    public SchoolSearchController() {
        this.schools = new HashMap<String, School>();
        School school = new School("NKU", "TJU");
        school.addMajor(new Major("Computer Science"));
        school.addMajor(new Major("Artificial Intelligence"));
        school.addMajor(new Major("Software Engineering"));
        school.addMentor(new Mentor("John Smith", "Computer Science", "Professor"));
        school.addMentor(new Mentor("Jane Doe", "Computer Science", "Associate
            Professor"));
        this.schools.put(school.getName(), school);
    }

    @GetMapping("/grades")
    public ArrayList<String> getGrades(@RequestParam String studentId) {
        ArrayList<String> grades = new ArrayList<String>();
        return grades;
    }

    @GetMapping("/interests")
    public ArrayList<String> getInterests(@RequestParam String studentId) {
        ArrayList<String> interests = new ArrayList<String>();
        interests.add("Reading");
        interests.add("Coding");
        interests.add("Sports");
        return interests;
    }

    @GetMapping("/career")
    public ArrayList<String> getCareer(@RequestParam String studentId) {
        ArrayList<String> career = new ArrayList<String>();
        career.add("Engineer");
        career.add("Doctor");
        career.add("Lawyer");
```

```
        return career;
    }

    @GetMapping("/school")
    public School getSchoolInfo(@RequestParam String schoolName) {
        if (this.schools.containsKey(schoolName)) {
            return this.schools.get(schoolName);
        } else {
            return null;
        }
    }
}

class School {
    private String name;
    private String location;
    private ArrayList<Major> majors;
    private ArrayList<Mentor> mentors;

    public School(String name, String location) {
        this.name = name;
        this.location = location;
        this.majors = new ArrayList<Major>();
        this.mentors = new ArrayList<Mentor>();
    }

    public String getName() {
        return name;
    }

    public String getLocation() {
        return location;
    }

    public void addMajor(Major major) {
        this.majors.add(major);
    }

    public ArrayList<Major> getMajors() {
        return majors;
    }
```

```
    public void addMentor(Mentor mentor) {
        this.mentors.add(mentor);
    }

    public ArrayList<Mentor> getMentors() {
        return mentors;
    }
}

class Major {
    private String name;

    public Major(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

class Mentor {
    private String name;
    private String field;
    private String title;

    public Mentor(String name, String field, String title) {
        this.name = name;
        this.field = field;
        this.title = title;
    }

    public String getName() {
        return name;
    }

    public String getField() {
        return field;
    }

    public String getTitle() {
        return title;
```

```
    }
}
```

5.3 管理员

```
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.HashMap;

@RestController
@RequestMapping("/admin")
public class SchoolAdminController {
    private HashMap<String, School> schools;

    public SchoolAdminController() {
        this.schools = new HashMap<String, School>();
        School school = new School("Harvard", "Cambridge, MA");
        school.addMajor(new Major("Computer Science"));
        school.addMajor(new Major("Biology"));
        school.addMajor(new Major("History"));
        school.addMentor(new Mentor("John Smith", "Computer Science",
            "Professor"));
        school.addMentor(new Mentor("Jane Doe", "Computer Science", "Associate
            Professor"));
        this.schools.put(school.getName(), school);
    }

    @PostMapping("/addSchool")
    public boolean addSchool(@RequestBody School school) {
        if (this.schools.containsKey(school.getName())) {
            return false;
        } else {
            this.schools.put(school.getName(), school);
            return true;
        }
    }

    @PutMapping("/updateSchool")
```

```
public boolean updateSchool(@RequestParam String schoolName, @RequestBody
    School school) {
    if (this.schools.containsKey(schoolName)) {
        this.schools.put(schoolName, school);
        return true;
    } else {
        return false;
    }
}

@DeleteMapping("/deleteSchool")
public boolean deleteSchool(@RequestParam String schoolName) {
    if (this.schools.containsKey(schoolName)) {
        this.schools.remove(schoolName);
        return true;
    } else {
        return false;
    }
}

@PostMapping("/addMajor")
public boolean addMajor(@RequestParam String schoolName, @RequestBody Major
    major) {
    if (this.schools.containsKey(schoolName)) {
        this.schools.get(schoolName).addMajor(major);
        return true;
    } else {
        return false;
    }
}

@PutMapping("/updateMajor")
public boolean updateMajor(@RequestParam String schoolName, @RequestParam
    String majorName, @RequestBody Major major) {
    if (this.schools.containsKey(schoolName)) {
        ArrayList<Major> majors = this.schools.get(schoolName).getMajors();
        for (int i = 0; i < majors.size(); i++) {
            if (majors.get(i).getName().equals(majorName)) {
                majors.set(i, major);
                return true;
            }
        }
```

```
            return false;
        } else {
            return false;
        }
    }

    @DeleteMapping("/deleteMajor")
    public boolean deleteMajor(@RequestParam String schoolName, @RequestParam
        String majorName) {
        if (this.schools.containsKey(schoolName)) {
            ArrayList<Major> majors = this.schools.get(schoolName).getMajors();
            for (int i = 0; i < majors.size(); i++) {
                if (majors.get(i).getName().equals(majorName)) {
                    majors.remove(i);
                    return true;
                }
            }
            return false;
        } else {
            return false;
        }
    }

    @PostMapping("/addMentor")
    public boolean addMentor(@RequestParam String schoolName, @RequestBody
        Mentor mentor) {
        if (this.schools.containsKey(schoolName)) {
           this.schools.get(schoolName).addMentor(mentor);
            return true;
        } else {
            return false;
        }
    }

    @PutMapping("/updateMentor")
    public boolean updateMentor(@RequestParam String schoolName, @RequestParam
        String mentorName, @RequestBody Mentor mentor) {
        if (this.schools.containsKey(schoolName)) {
            ArrayList<Mentor> mentors = this.schools.get(schoolName).getMentors();
            for (int i = 0; i < mentors.size(); i++) {
                if (mentors.get(i).getName().equals(mentorName)) {
                    mentors.set(i, mentor);
```

```
                return true;
            }
        }
        return false;
    } else {
        return false;
    }
}

@DeleteMapping("/deleteMentor")
public boolean deleteMentor(@RequestParam String schoolName, @RequestParam
    String mentorName) {
    if (this.schools.containsKey(schoolName)) {
        ArrayList<Mentor> mentors = this.schools.get(schoolName).getMentors();
        for (int i = 0; i < mentors.size(); i++) {
            if (mentors.get(i).getName().equals(mentorName)) {
                mentors.remove(i);
                return true;
            }
        }
        return false;
    } else {
        return false;
    }
}

@GetMapping("/getSchools")
public ArrayList<School> getSchools() {
    return new ArrayList<School>(this.schools.values());
}

@GetMapping("/getSchool")
public School getSchool(@RequestParam String schoolName) {
    return this.schools.get(schoolName);
}

@GetMapping("/getMajors")
public ArrayList<Major> getMajors(@RequestParam String schoolName) {
    return this.schools.get(schoolName).getMajors();
}

@GetMapping("/getMajor")
```

```
    public Major getMajor(@RequestParam String schoolName, @RequestParam String
        majorName) {
        ArrayList<Major> majors = this.schools.get(schoolName).getMajors();
        for (int i = 0; i < majors.size(); i++) {
            if (majors.get(i).getName().equals(majorName)) {
                return majors.get(i);
            }
        }
        return null;
    }

    @GetMapping("/getMentors")
    public ArrayList<Mentor> getMentors(@RequestParam String schoolName) {
        return this.schools.get(schoolName).getMentors();
    }

    @GetMapping("/getMentor")
    public Mentor getMentor(@RequestParam String schoolName, @RequestParam
        String mentorName) {
        ArrayList<Mentor> mentors = this.schools.get(schoolName).getMentors();
        for (int i = 0; i < mentors.size(); i++) {
            if (mentors.get(i).getName().equals(mentorName)) {
                return mentors.get(i);
            }
        }
        return null;
    }
}
```

5.4 加密算法

5.4.1 AES

高级加密标准(Advanced Encryption Standard, AES)，对称加密算法。用于对数据的加密存储工作。

```
package AES

import (
   "bytes"
   "crypto/aes"
```

```
    "crypto/cipher"
    "fmt"
    uuid "github.com/satori/go.uuid"
)

func GenerateRandomKeyOfAES() []byte {
    return []byte(uuid.Must(uuid.NewV4()).String()[:16])
}

func decryptAES(cipherText, key []byte) []byte {
    block, _ := aes.NewCipher(key)
    blockSize := block.BlockSize()
    blockMode := cipher.NewCBCDecrypter(block, key[:blockSize])
    originData := make([]byte, len(cipherText))
    blockMode.CryptBlocks(originData, cipherText)
    originData = PKCS7UnPadding(originData)
    return originData
}

func PKCS7UnPadding(originData []byte) []byte {
    length := len(originData)
    position := int(originData[length-1])
    return originData[:length-position]
}

func encryptAES(originData, key []byte) []byte {
    block, _ := aes.NewCipher(key)
    originData = PKCS7Padding(originData, block.BlockSize())
    blockMode := cipher.NewCBCEncrypter(block, key[:block.BlockSize()])
    cipherText := make([]byte, len(originData))
    blockMode.CryptBlocks(cipherText, originData)
    return cipherText
}

func PKCS7Padding(originData []byte, blockSize int) []byte {
    padding := blockSize - len(originData)%blockSize
    padText := bytes.Repeat([]byte{byte(padding)}, padding)
    return append(originData, padText...)
}
```

5.4.2 ECC

椭圆曲线加密算法(Elliptic Curve Cryptography, ECC)，非对称加密算法。用于对文本类型密钥数据的加密存储工作。在此处使用了以太坊的ECIES加密方法实现ECC加密。

```
package ECC

import (
   "crypto/ecdsa"
   "crypto/elliptic"
   "crypto/rand"
   "crypto/x509"
   "encoding/hex"
   "encoding/pem"
   "fmt"
   "github.com/ethereum/go-ethereum/crypto/ecies"
   uuid "github.com/satori/go.uuid"
   mathRand "math/rand"
   "os"
   "strings"
   "time"
)

func GenerateRandomKeyOfECC() (*ecies.PrivateKey, *ecies.PublicKey, error) {
   randKey := createRandomSalt(55)
   privateKey, err := ecdsa.GenerateKey(elliptic.P256(), strings.NewReader(randKey))
   if err != nil {
      return nil, nil, err
   }
   privateBytes, err := x509.MarshalECPrivateKey(privateKey)
   if err != nil {
      return nil, nil, err
   }
   privateBlock := pem.Block{
      Type: "ecc private key",
      Bytes: privateBytes,
   }
   var path string
   path = "./"
   path += uuid.Must(uuid.NewV4()).String()
   path += "-private.pem"
   privateFileOutput, err := os.Create(path)
```

```
if err != nil {
   return nil, nil, err
}
defer privateFileOutput.Close()
err = pem.Encode(privateFileOutput, &privateBlock)
if err != nil {
   return nil, nil, err
}
privateFileInput, err := os.Open(path)
if err != nil {
   return nil, nil, err
}
defer privateFileInput.Close()
privateKeyContent, err := privateFileInput.Stat()
if err != nil {
   return nil, nil, err
}
privateKeyBuffer := make([]byte, privateKeyContent.Size())
_, err = privateFileInput.Read(privateKeyBuffer)
if err != nil {
   return nil, nil, err
}
privateReaderBlock, _ := pem.Decode(privateKeyBuffer)
privateKeyBytes, err := x509.ParseECPrivateKey(privateReaderBlock.Bytes)
if err != nil {
   return nil, nil, err
}
privateKeyECIES := ecies.ImportECDSA(privateKeyBytes)
go RemovePemFile(path)
publicKey := privateKey.PublicKey
publicBytes, err := x509.MarshalPKIXPublicKey(&publicKey)
if err != nil {
   return nil, nil, err
}
publicBlock := pem.Block{
   Type: "ecc public key",
   Bytes: publicBytes,
}
path = "./"
path += uuid.Must(uuid.NewV4()).String()
path += "-public.pem"
publicFileOutput, err := os.Create(path)
```

```
    if err != nil {
        return nil, nil, err
    }
    defer publicFileOutput.Close()
    err = pem.Encode(publicFileOutput, &publicBlock)
    if err != nil {
        return nil, nil, err
    }
    publicFileInput, err := os.Open(path)
    if err != nil {
        return nil, nil, err
    }
    defer publicFileInput.Close()
    publicKeyContent, err := publicFileInput.Stat()
    if err != nil {
        return nil, nil, err
    }
    publicKeyBuffer := make([]byte, publicKeyContent.Size())
    _, err = publicFileInput.Read(publicKeyBuffer)
    if err != nil {
        return nil, nil, err
    }
    publicReaderBlock, _ := pem.Decode(publicKeyBuffer)
    publicKeyBytes, err := x509.ParsePKIXPublicKey(publicReaderBlock.Bytes)
    if err != nil {
        return nil, nil, err
    }
    publicKeyInner := publicKeyBytes.(*ecdsa.PublicKey)
    publicKeyECIES := ecies.ImportECDSAPublic(publicKeyInner)
    go RemovePemFile(path)
    return privateKeyECIES, publicKeyECIES, nil
}

func EncryptECC(srcData string, publicKey *ecies.PublicKey) (cryptData string, err
    error) {
    encryptBytes, err := ecies.Encrypt(rand.Reader, publicKey, []byte(srcData), nil, nil)
    if err != nil {
        return "", err
    }
    cryptData = hex.EncodeToString(encryptBytes)
    return
}
```

```
func DecryptECC(cryptData string, privateKey *ecies.PrivateKey) (srcData string, err
    error) {
   cryptBytes, err := hex.DecodeString(cryptData)
   srcByte, err := privateKey.Decrypt(cryptBytes, nil, nil)
   if err != nil {
      return "", err
   }
   srcData = string(srcByte)
   return
}
```

5.5 http通信中间件

```
package pkg

import (
    "encoding/json"
    "errors"
    "fmt"
    "net"
    "net/http"
    "net/url"
    "strconv"
    "strings"
)

const (
    HttpResponseTrue = 0
    HttpResponseFalse = 1
)

const OKString = "OK."

func GetRealIpAddress(r *http.Request) (string, error) {
    xForwardedFor := r.Header.Get("X-Forwarded-For")
    ip := strings.TrimSpace(strings.Split(xForwardedFor, ",")[0])
    if "" != ip {
        return ip, nil
    }
```

```
    ip = strings.TrimSpace(r.Header.Get("X-Real-Ip"))
    if "" != ip {
        return ip, nil
    }

    ip, _, err := net.SplitHostPort(strings.TrimSpace(r.RemoteAddr))
    return ip, err
    // return "", errors.New("Failed to get ip address. ")
}

type JsonResponse struct {
    Status int             `json:"status"`
    Message json.RawMessage `json:"message"`
    Code   int             `json:"code"`
}

func SendJsonResponse(w http.ResponseWriter, r *http.Request, response
     JsonResponse) {
    result, err := json.Marshal(response)
    if nil != err {
        fmt.Println(err.Error())
    }
    w.Header().Set("content-type", "application/json")
    w.WriteHeader(response.Code)
    _, err = w.Write(result)
    if nil != err {
        fmt.Println(err.Error())
    }
}

func DealJsonMarshal(w http.ResponseWriter, r *http.Request, v interface{})
     []byte {
    data, err := json.Marshal(v)
    if nil != err {
        SendStringResponse(w, r, StringResponse{
            Status: HttpResponseFalse,
            Message: "Server Error. Json: " + err.Error(),
            Code:   http.StatusInternalServerError,
        })
        return nil
    }
```

```
    return data
}

func WriteJsonMarshal(w http.ResponseWriter, r *http.Request, v interface{}) {
    message := DealJsonMarshal(w, r, v)
    if nil != message {
        SendJsonResponse(w, r, JsonResponse{
            Status: HttpResponseTrue,
            Message: message,
            Code:   http.StatusOK,
        })
    }
}

type StringResponse struct {
    Status int    `json:"status"`
    Message string `json:"message"`
    Code   int    `json:"code"`
}

func makeResponseOK() StringResponse {
    return StringResponse{
        Status: HttpResponseTrue,
        Message: OKString,
        Code:   http.StatusOK,
    }
}

func (res StringResponse) Send(w http.ResponseWriter, r *http.Request) {
    result, err := json.Marshal(res)
    if nil != err {
        fmt.Println(err)
        return
    }
    SetContentJsonHeader(w)
    w.WriteHeader(res.Code)
    _, err = w.Write(result)
    if nil != err {
        fmt.Println(err)
    }
}
```

```
func SendStringResponse(w http.ResponseWriter, r *http.Request, response
    StringResponse) {
    result, err := json.Marshal(response)
    if nil != err {
        fmt.Println(err.Error())
    }
    w.Header().Set("content-type", "application/json")
    w.WriteHeader(response.Code)
    _, err = w.Write(result)
    if nil != err {
        fmt.Println(err.Error())
    }
}

func SendResponseOK(w http.ResponseWriter, r *http.Request) {
    makeResponseOK().Send(w, r)
}

func SendResponseInternalError(w http.ResponseWriter, r *http.Request, err
    error, code int) {
    StringResponse{
        Status: code,
        Message: err.Error(),
        Code:   http.StatusInternalServerError,
    }.Send(w, r)
}

type Method struct {
    Pattern string                                          `json:"pattern"`
    Handler func(writer http.ResponseWriter, request *http.Request)
        `json:"handler"`
}

func MethodGenerator(pattern string, handler func(writer http.ResponseWriter,
    request *http.Request)) Method {
    return Method{
        Pattern: pattern,
        Handler: handler,
    }
}

type Router struct {
```

```
    Name    string  `json:"name"`
    Mask    string  `json:"mask"`
    Methods []Method `json:"methods"`
}

func (router Router) attachServer(sm *http.ServeMux, mask string) {
    for i := range router.Methods {
        // log.Println(mask + router.Mask + router.Methods[i].Pattern) // TODO
            Remove router output.
        sm.HandleFunc(mask+router.Mask+router.Methods[i].Pattern,
            router.Methods[i].Handler)
    }
}

func SetDefaultHeaders(w http.ResponseWriter) {

}

func SetContentJsonHeader(w http.ResponseWriter) {
    w.Header().Set("content-type", "application/json")
}

type Server struct {
    Path    string  `json:"path"`
    Port    int     `json:"port"`
    Mask    string  `json:"mask"`
    Routers []Router `json:"routers"`
}

func (server Server) Listen() {
    sm := http.NewServeMux()
    for i := range server.Routers {
        server.Routers[i].attachServer(sm, server.Mask)
    }
    err := MuxListen(server.Path, server.Port, sm)
    if nil != err {
        fmt.Println(err.Error())
    }
}

func MuxListen(path string, port int, mux *http.ServeMux) error {
    return http.ListenAndServe(path+":"+strconv.Itoa(port), mux)
```

```
}

func CheckNeedArgs(values url.Values, args ...string) error {
    message := ""
    for _, value := range args {
        if !values.Has(value) {
            message += value + ", "
        }
    }
    if "" != message {
        return errors.New(message + "arg(s) is(are) not defined.")
    }
    return nil
}

type GetRequest struct {
    Query url.Values `json:"query"`
    Err   error      `json:"err"`
}

func DealGetRequest(w http.ResponseWriter, r *http.Request, args ...string)
     GetRequest {
    SetDefaultHeaders(w)
    request := GetRequest{
        Query: r.URL.Query(),
        Err:   nil,
    }
    if "GET" != r.Method {
        if "OPTIONS" != r.Method {
            request.Err = errors.New("Request Must be 'Get'. ")
            StringResponse{
                Status: HttpResponseFalse,
                Message: "Server Error. " + request.Err.Error(),
                Code:   http.StatusMethodNotAllowed,
            }.Send(w, r)
        } else {
            request.Err = errors.New("OPTIONS")
            SendResponseOK(w, r)
        }
        return request
    }
    request.Err = CheckNeedArgs(request.Query, args...)
```

```
    if nil != request.Err {
        StringResponse{
            Status: HttpResponseFalse,
            Message: "Server Error. " + request.Err.Error(),
            Code:   http.StatusExpectationFailed,
        }.Send(w, r)
    }
    return request
}

type PostRequest struct {
    Err error `json:"err"`
}

func DealPostRequest(w http.ResponseWriter, r *http.Request, args ...string)
     PostRequest {
    SetDefaultHeaders(w)
    request := PostRequest{
        Err: r.ParseForm(),
    }
    if "POST" != r.Method {
        if "OPTIONS" != r.Method {
            request.Err = errors.New("Request Must be 'Post'. ")
            StringResponse{
                Status: HttpResponseFalse,
                Message: "Server Error. " + request.Err.Error(),
                Code:   http.StatusMethodNotAllowed,
            }.Send(w, r)
        } else {
            request.Err = errors.New("OPTIONS")
            SendResponseOK(w, r)
        }
        return request
    }
    request.Err = CheckNeedArgs(r.Form, args...)
    if nil != request.Err {
        StringResponse{
            Status: HttpResponseFalse,
            Message: "Server Error. " + request.Err.Error(),
            Code:   http.StatusExpectationFailed,
        }.Send(w, r)
    }
```

```
    return request
}

func DealPostError() {}

type Dealer struct {
    Header  func(w http.ResponseWriter) `json:"header"`
    Handlers []Handler                `json:"handlers"`
}

type Handler interface {
    Dealer(w http.ResponseWriter, r *http.Request) (http.ResponseWriter,
        *http.Request, error)
}

func (dealer Dealer) Deal(w http.ResponseWriter, r *http.Request)
    (http.ResponseWriter, *http.Request, error) {
    if nil != dealer.Header {
        dealer.Header(w)
    }
    wc, rc := w, r
    var err error
    for i := range dealer.Handlers {
        if nil == dealer.Handlers[i].Dealer {
            continue
        }
        wc, rc, err = dealer.Handlers[i].Dealer(wc, rc)
        if err != nil {
            return w, r, err
        }
    }
    return wc, rc, nil
}
```