同济大学

TONGJI UNIVERSITY

《WEB 技术》

实验报告

实验名称	实验 4 服务端编程
小组成员	郑博远（2154312）
学院（系）	电子与信息工程学院计算机科学与技术系
专　　业	计算机科学与技术
任课教师	郭玉臣
日　　期	2023 年 5 月 15 日

一、实验原理

本次实验基于实验 3 中前端实现的媒体播放器，使用 Flask 框架、SQLite 数据库以及相应拓展实现了服务端编程。在本次实验中，媒体播放器网站将增添用户身份认证、上传视频播放列表、管理员编辑播放列表、管理员下载视频等功能。实验将使用 HTML、CSS 和原生 JavaScript 来实现前端界面和媒体播放器的功能，并使用 Flask 作为后端框架来处理用户认证、数据存储和提供 API 接口，从而构建一个具备身份认证和管理功能的媒体播放器应用程序。本实验的实验原理如下：

1.1 Flask 框架

Flask 是一个常用的 Python Web 框架，其简单而灵活的特点允许开发人员快速构建具有动态功能的 Web 应用程序。Flask 的核心是 WSGI（Web Server Gateway Interface）应用程序，它将 HTTP 请求从 Web 服务器传递给应用程序，然后将响应返回给服务器。Flask 还提供了许多扩展，以便于处理常见的 Web 开发任务。

Flask 的工作原理如下：

1. 路由和视图函数：Flask 通过使用路由定义来映射 URL 到相应的视图函数。当用户请求特定的 URL 时，Flask 将调用相应的视图函数来处理请求并生成响应。

2. 请求-响应循环：当接收到 HTTP 请求时，Flask 会创建一个请求上下文（request context），在该上下文中可以访问请求相关的数据，如请求参数、头部信息等。然后 Flask 将请求传递给相应的视图函数进行处理。视图函数生成一个响应，然后 Flask 将响应返回给客户端。

3. 模板引擎：Flask 使用模板引擎（如 Jinja2）来生成动态内容。模板引擎允许在 HTML 中嵌入 Python 代码，并根据需要将其渲染为静态 HTML。这样可以实现页面的动态生成。

1.2 SQLite 数据库

SQLite 是一个轻量级的嵌入式关系型数据库管理系统，它在本地文件中存储数

据，并提供了相应 SQL 操作用于管理数据。与传统的数据库管理系统相比，SQLite 不需要独立的服务器进程，而是直接嵌入到应用程序中，简化了部署和管理的过程。

SQLite 的工作原理如下：

1. 数据库文件：SQLite 使用一个单一的数据库文件来存储所有数据。该文件通常具有.db 或.sqlite 扩展名。可以使用 SQLite 命令行工具或支持 SQLite 的库来管理和访问数据库文件。

2. 数据表：在 SQLite 中，数据存储在表中。表由列和行组成，类似于传统的关系型数据库。每个表都有一个唯一的名称，并定义了每列的数据类型和约束条件。

3. SQL 操作：SQLite 支持标准的 SQL 操作，包括查询（SELECT）、插入（INSERT）、更新（UPDATE）和删除（DELETE）等。开发人员可以使用这些 SQL 语句来操作数据库中的数据。

4. 数据库连接：应用程序可以通过 SQLite 的 API 连接到数据库文件，并执行相应的 SQL 操作。连接对象允许应用程序与数据库通信并管理事务的提交和回滚。

5. 事务支持：SQLite 支持事务的概念，可以确保一系列操作要么全部执行成功，要么全部回滚。事务可以保证数据的完整性和一致性。

6. 索引和查询优化：SQLite 支持创建索引来提高查询性能。索引是一种数据结构，可以加速数据检索操作。SQLite 还提供了查询优化器，它可以分析查询语句并选择最优的执行计划。

1.3 Flask-SQLAlchemy

Flask-SQLAlchemy 是基于 SQLAlchemy 的 Python SQL 工具包，其扩展简化了与关系型数据库的交互，提供了模型定义、数据库会话和查询构建等功能，方便开发人员进行数据库操作。这些扩展的结合使用，可以帮助开发人员快速搭建具有认证功能和数据库支持的 Web 应用

Flask-SQLAlchemy 的工作原理如下：

1. 定义模型：开发人员可以使用 Python 类来定义数据模型，每个类对应数据

库中的一个表。模型类的属性对应表的列，通过定义模型类，可以轻松地创建、更新和查询数据库表。

2. 数据库会话：Flask-SQLAlchemy 提供了一个数据库会话（Session）对象，用于与数据库进行交互。会话对象允许开发人员执行数据库查询、插入、更新和删除操作。

3. 查询构建器：Flask-SQLAlchemy 提供了一种方便的方式来构建和执行 SQL 查询，而无需直接编写 SQL 语句。它使用方法链式调用的方式来构建查询，包括选择、过滤、排序等操作。

4. 关系映射：Flask-SQLAlchemy 支持对象关系映射（ORM），可以将数据库表和模型类之间建立映射关系。这样，开发人员可以通过操作模型类来实现对数据库的操作，而不需要直接编写 SQL 语句。

装

1.4 Flask-Login

订

Flask-Login 是 Flask 的另一个扩展，可用于处理用户认证和管理用户会话。它简化了用户登录、登出以及对受保护页面的访问控制等操作。

Flask-Login 的工作原理如下：

线

1. 用户认证：当用户进行登录操作时，Flask-Login 提供了一个函数来验证用户提供的凭据（如用户名和密码）。开发人员可以实现自定义的用户认证逻辑，以满足特定的应用程序需求。

2. 用户会话管理：一旦用户登录成功，Flask-Login 会将用户信息存储在会话中。在后续的请求中，Flask-Login 可以根据会话中的信息来识别当前登录的用户。

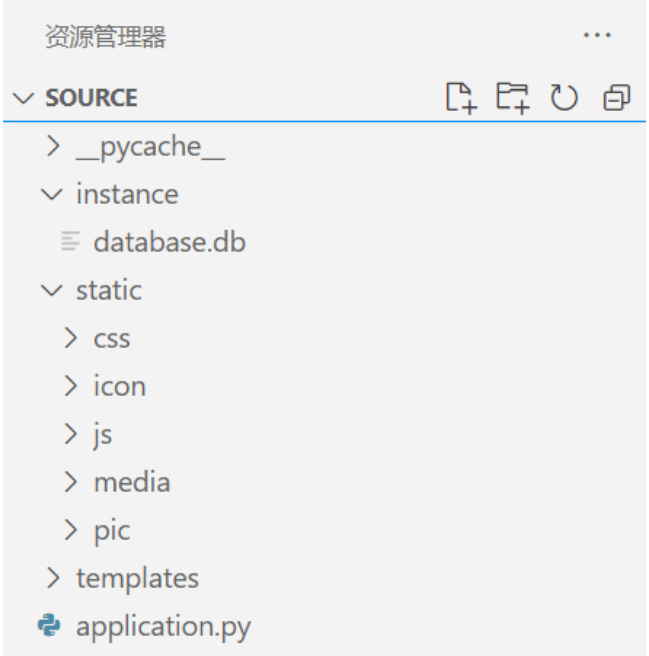
3. 访问控制：Flask-Login 还提供了装饰器@login_required，用于限制只有登录用户才能访问的页面。未登录的用户会被重定向到登录页面并收到错误提示。

二、实验步骤

本次实验在实验 3 的基础上实现了包含服务端编程的 Web 播放器。通过前端页

面中的媒体列表和媒体播放器，用户可以浏览、播放和上传媒体文件。前端通过与后端的交互，实现了获取媒体列表、下载媒体文件（管理员权限）和编辑媒体列表的功能（管理员权限）。后端接收前端的请求后，在验证用户身份和登录状态后执行查询数据库并处理相关操作，将数据和文件传递给前端，实现了媒体管理和媒体文件的上传、下载和删除功能。在本次实验之前，我还没有接触过服务端编程的相关知识；因此在本次实验中，我从阅读文档开始从零学习了 Flask 有关的后端相关知识并加以运用。下面将分点介绍本次实验的主要实验步骤。

2.1 项目结构搭建



实验开始之前，我首先进行了实验的项目结构搭建。项目的根目录下共有 instance、static、templates 三个文件夹；此外，还有 application.py 这一 Python 文件，其中涵盖 Flask 后端的相关 Python 代码。在 instance 文件夹下，存放了 SQLite 数据库的.db 文件；在 templates 文件夹下，存放了 home.html、player.html 等 html 模板文件；static 文件夹用于存放静态文件，其目录下又分为 css、icon、js、media、pic 五个文件夹。其中 css 与 js 文件夹分别存放前端所需的层叠样式表文件和 JavaScript 文件，icon 文件夹存放了用于显示的图标文件（.svg），media 文件夹存放了用户上传的视频或音频等媒体文件，pic 文件夹存放媒体的缩略图。

2.2 网站页面划分

在前端设计部分，我在实验 3 的基础上针对新的功能增加了几个页面，着重进行了以下页面的设计：登录页面、注册页面、主页、播放器页面、上传页面。

登录页面：该页面用于用户登录，包含用户名和密码的输入框，以及登录按钮。用户可以输入正确的用户名和密码进行登录，或选择注册按钮跳转到注册页面。

注册页面：用户可以在该页面进行新用户的注册，需要填写用户名、密码和电子邮件，并提交注册表单。注册成功后，会自动登录，并跳转到主页。在实际实现是，由于这两个页面功能都较少，我合并到了一个 html 文件中，通过按钮的点击配合 CSS 动画实现登录和页面界面的效果跳转。

主页：该页面主要展示了网站的主要功能和理念，起到门户的作用。点击“即可开启”字样可以跳转到播放器页面。若用户暂未登录，则会被强制重定向至登录、注册页面进行账号的登录。

播放器页面：当用户登录之后，网站会自动跳转到播放器页面。该页面与实验 3 中相同，包含一个自制的媒体播放器，同时可以在播放列表中选择媒体源进行播放。列表中罗列了媒体的标题、作者等相关信息。用户可以通过播放器来观看视频或听取音频，管理员还有权限删除视频或者下载视频到本地。

上传页面：登录后的用户可以访问上传页面，在该页面可以选择要上传的媒体文件和对应的缩略图文件。用户需要填写媒体的标题和副标题，并提交上传表单。上传成功后，会自动刷新页面并显示最新的媒体列表。

2.3 数据库搭建

在实验中，我使用了 SQLite 数据库进行数据存储。通过 Flask 框架提供的 SQLAlchemy 扩展，能够轻松地进行数据库的连接和操作。

首先，我在应用程序的配置中指定了 SQLite 数据库的 URI，即 app.config["SQLALCHEMY_DATABASE_URI"]。这样，Flask 就知道要连接的数据库是 SQLite，并且指定了数据库文件的路径。

接下来，我定义了两个数据表：Users 和 Media，分别用于存储用户信息和媒体信息。在 Users 表中，我定义了几个字段：id、username、password、email 和 admin。其中，id 是主键字段，username、password、email 和 admin 分别表示用户名、密码、电子邮箱和管理员标识。在 Media 表中，我类似地定义了字段：id、title、subtitle、picSrc 和 mediaSrc，用于存储标题、副标题、缩略图路径和媒体路径等信息。

通过在这两个类中定义字段和相关的属性和方法，便能够方便地操作数据库。例如，在 Users 类中，我定义了__repr__方法，用于在调试和输出时显示用户对象的有意义的表示。而在 Media 类中，我还定义了 to_json 方法，用于将媒体对象转换为 JSON 格式的字典，方便在 API 接口中返回媒体信息。

为了实现数据库的初始化和管理，我使用了 db.init_app(app)来将数据库与应用程序关联起来。这样，我就能够使用 db 对象来执行数据库相关的操作，包括创建表、插入数据、查询数据等。以上步骤成功搭建了 SQLite 数据库，并定义了两个数据表以存储用户信息和媒体信息；这为后续的用户注册、登录、媒体管理等功能提供了数据存储的基础。

装 订 线

2.4 Flask-login 基础上的身份认证

在实验中，为了实现用户登录和身份认证功能，我选择使用了 Flask-Login 扩展库。首先，在后端的 application.py 文件中，我导入了 flask_login 模块，并创建了一个 LoginManager 对象，用于管理用户登录和会话信息。

为了与 Flask-Login 集成，我创建了一个名为 User 的类，它继承自 UserMixin。UserMixin 是 Flask-Login 提供的一个基类，提供了一些常用的用户相关方法的实现。通过继承 UserMixin，User 类获得了登录状态管理、会话管理等功能。

之后，我实现了 login_manager.user_loader 和 login_manager.request_loader 两个函数。这两个函数是 Flask-Login 提供的回调函数，用于加载用户信息和处理用户请求。login_manager.user_loader 函数用于通过用户 ID 加载用户信息。在这个函数中，我查询数据库，根据用户 ID 找到相应的用户对象，并返回该用户对象。这样，Flask-Login 就能根据用户 ID 来恢复用户的会话信息。login_manager.request_loader

函数用于通过请求加载用户信息。在这个函数中，我从请求中获取用户名，然后查询数据库，根据用户名找到相应的用户对象，并返回该用户对象。这样，Flask-Login 就能根据请求中的用户名来恢复用户的会话信息。

在登录页面的路由处理函数 login 中，我使用了 request.method 来判断请求的方法。当请求方法为 GET 时，表示用户正在访问登录页面，我渲染了登录页面的模板。当请求方法为 POST 时，表示用户提交了登录表单，我通过查询数据库验证用户的用户名和密码是否匹配。如果验证通过，我创建了一个 User 对象，将用户名作为其 ID，并调用 login_user 函数将用户对象添加到当前会话中，实现用户的登录操作。

为了确保只有已登录的用户才能访问需要身份认证的页面，我使用了 @login_required 装饰器。将这个装饰器应用在需要进行身份认证的路由处理函数上，当用户未登录时，Flask-Login 会自动将用户重定向到登录页面。

对于需要管理员权限的操作，我仿照 @login_required 自己定义了一个 @admin_required 装饰器函数。这个装饰器函数用于检查当前用户是否具有管理员权限。在装饰器函数内部，我查询数据库，获取当前用户的权限信息，并在数据库中查询信息以判断其是否为管理员。如果当前用户不是管理员，我返回一个包含错误信息的 JSON 响应，并设置 HTTP 状态码为 403，表示权限不足。

在实验中，我还为登录和权限验证过程提供了相应的错误提示。通过设置 login_manager.login_message 和 login_manager.login_message_category，我定义了登录时的提示消息和消息的分类。当用户访问需要登录才能访问的页面时，如果未登录，Flask-Login 会自动将用户重定向到登录页面，并显示设置的登录提示消息。

2.5 前后端交互

在这一部分，我将详细描述前后端之间的交互过程，包括登录表单信息上传、媒体源列表获取、视频上传以及视频下载。

1. 登录表单信息上传

当用户填写登录表单并点击提交按钮时，前端会将表单中的用户名和密码信息

通过 POST 请求发送到后端的登录接口。需要特别注意的是，由于用户密码是敏感信息，不适合明文存储在数据库中。前端会先对用户的密码进行 MD5 加密，再发送到后端进行校验。后端的 Flask 路由 /login 负责处理该请求。

在后端的处理过程中，首先从请求的表单数据中获取到用户名和 md5 加密后的密码。然后，在数据库中查询是否存在该用户名的用户，并验证输入的密码是否正确。如果用户名和密码验证通过，则创建一个新的用户对象，并使用 Flask-Login 提供的 login_user 函数进行登录，将用户信息存储在会话中。

如果用户名或密码验证失败，我将使用 Flask 提供的 flash 函数向用户显示错误消息，并重定向回登录页面，提醒用户重新输入正确的信息。

2. 媒体源列表获取

在用户成功登录后，用户将被重定向到媒体播放页面 /player。在该页面加载时，前端会发送一个 GET 请求到后端的媒体列表获取接口 /player/getMediaList。特别注意的是，该接口被@login_required 修饰，要求用户需登录后才能访问。

后端的 Flask 路由 /player/getMediaList 负责处理该请求。在处理过程中，通过查询数据库中的媒体表 Media，获取所有媒体的信息。然后，我将媒体信息转换为 JSON 格式，并作为响应返回给前端。前端收到响应后，可以解析 JSON 数据，并使用这些媒体信息在页面上动态生成媒体列表，以供用户选择和播放。

3. 视频上传

前端通过使用 HTML 表单来实现视频上传功能。用户可以在表单中填写标题和副标题，并选择要上传的缩略图和媒体文件。

后端的 Flask 路由 /upload 负责处理视频上传请求。在该路由中，首先判断请求的方法是 GET 还是 POST。如果是 GET 方法，表示用户刚进入上传页面，则通过渲染模板文件 upload.html 来展示上传表单给用户。如果是 POST 方法，表示用户提交了上传表单。首先从请求中获取上传的文件对象，通过检查文件对象是否存在以及其对应的字段名，判断用户是否上传了必要的缩略图和媒体文件。

如果缩略图或媒体文件缺失，我使用 Flask 提供的 flash 函数向用户显示错误消息，并重定向回上传页面，提醒用户重新选择并上传缺失的文件。如果缩略图和媒体文件都存在，则使用 secure_filename 函数对文件名进行安全处理，然后将文件保存到服务器的指定目录中。

接着，我将文件的保存路径和用户填写的标题、副标题等信息组合成一个新的媒体对象，并将其添加到数据库中的媒体表 Media。最后，通过提交数据库会话的更改，将新媒体对象保存到数据库中。

最后，将重定向用户到媒体播放页面，以便浏览新上传的媒体内容。

4. 视频下载

在媒体播放页面中，用户可以选择下载某个特定媒体的视频。当用户点击下载按钮时，前端会发送一个 GET 请求到后端的视频下载接口 /download/<int:media_id>，其中 media_id 是要下载的媒体的唯一标识符。特别注意的是，该接口不仅被@login_required 修饰，要求用户需登录后才能访问；还被我自定义的@admin_required 修饰，若用户不是管理员则将返回 JSON 错误提示信息，以及错误的 HTTP 状态码 403。

后端的 Flask 路由 /download/<int:media_id> 负责处理该请求。在处理过程中，我首先根据 media_id 查询数据库，获取对应媒体的信息。然后，我使用 Flask 提供的 send_from_directory 函数，从服务器的媒体存储目录中读取该媒体文件，并将其作为附件形式返回给用户。在请求成功的情况下（状态码为 200），前端会获取服务器返回的 Blob 数据，创建一个下载链接，并设置链接的属性，如链接地址和下载的文件名。然后，将链接添加到文档中，并模拟用户点击链接触发下载操作。如果请求失败或返回的是 JSON 数据（表示下载出现错误），前端会将返回的数据解析，并显示警告提示信息，提醒用户下载出现问题。

通过以上四个个交互过程，实现了用户登录表单信息上传、媒体源列表获取、视频上传和视频下载功能。这样，用户可以通过登录来访问媒体播放页面，并在页面上浏览媒体列表，选择播放或下载感兴趣的媒体内容。

三、实验内容

3.1 效果展示

运行 Flask 后，在浏览器中访问 http://127.0.0.1:5000，便能打开如下 home 主页面。在主页面中点击“即可开启”字样，若用户已经成功登录，则会跳转到播放器页面；若用户还未登录，则会跳转到登录注册页面。

上图展示的是登陆注册页面，对应的 url 为 http://127.0.1:5000/login。该页面包含注册和登录两个表单，用户可以通过填写用户名、密码和邮箱进行注册和登录操作。以下是页面的功能介绍：

登录表单：

· 用户需要在用户名和密码字段中填写登录信息。

· 点击登录按钮后，登录表单的提交会发送到后端的 /login 路由进行处理。

· 如果用户未填写必要的信息或账号密码错误，后端会将相应的错误消息通过 Flask 的 flash 函数传递到前端，并在页面中显示提示信息。如下图所示：

在登录或注册界面时，都有遮罩使得当前只显示对应内容。可以点击遮罩上的按钮进行“登录”与“注册”的切换。当用户点击 “已有帐号？”的登录按钮时，页面会切换到登录面板，显示登录表单。当用户点击 “还未注册？”的注册按钮时，页面会切换到注册面板，显示注册表单。

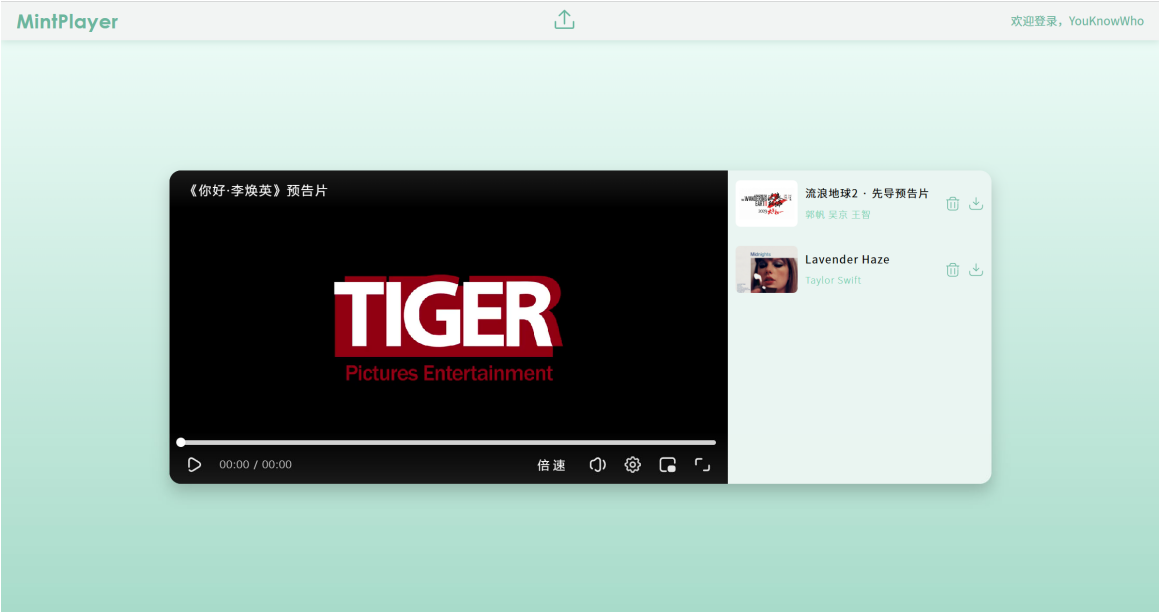
注册表单：

· 用户需要在用户名、密码和邮箱字段中填写相应的信息。

· 点击注册按钮后，注册表单的提交会发送到后端的 /signup 路由进行处理。

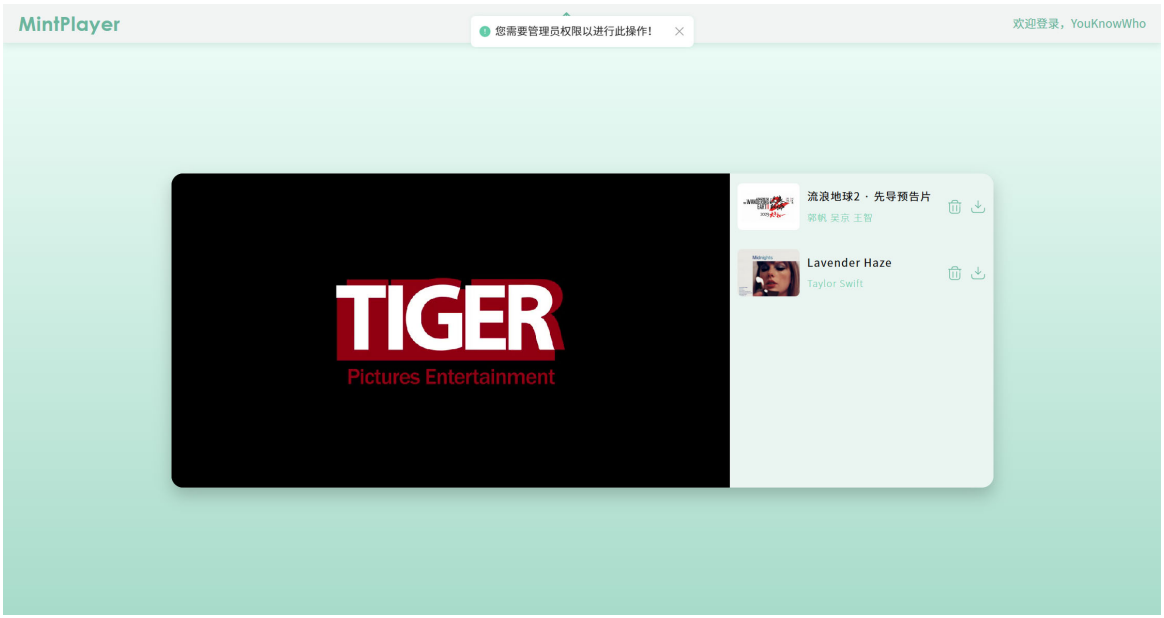
装 订 线

·如果用户未填写必要的信息或注册过程中出现错误，后端会将相应的错误消息通过 Flask 的 flash 函数传递到前端，并在页面中显示提示信息。

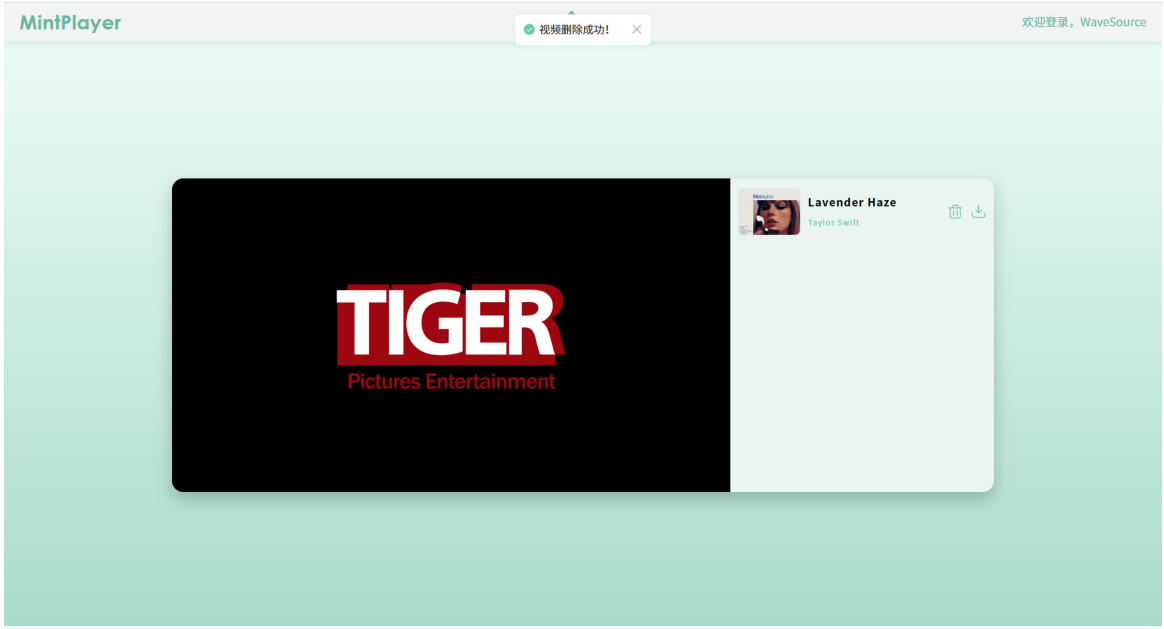


登录成功后，将会跳转到播放器页面，对应 url 为 http://127.0.1:5000/player。该页面有@login_required 装饰器修饰，因此如果未经注册直接访问则会被重定向到登录注册页面并给出错误提示信息。播放器页面如下图所示。屏幕中间是播放器主体，在鼠标悬停时能够展示出进度条、音量、播放暂停按钮等功能键；播放器的右侧是播放列表，展示了可供播放媒体的缩略图、标题、副标题等信息。每个列表对象右

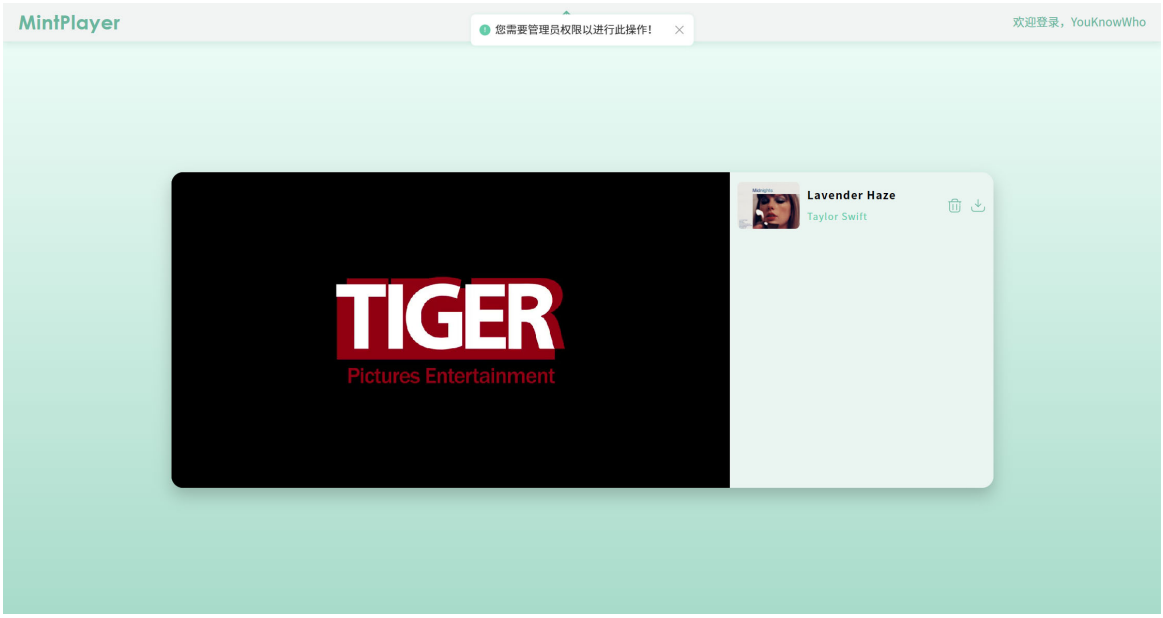
侧都有下载与删除按钮，拥有管理员权限的用户可以对列表进行编辑，或下载视频到本地。页面上方的中间是上传媒体按钮，点击可以到媒体上

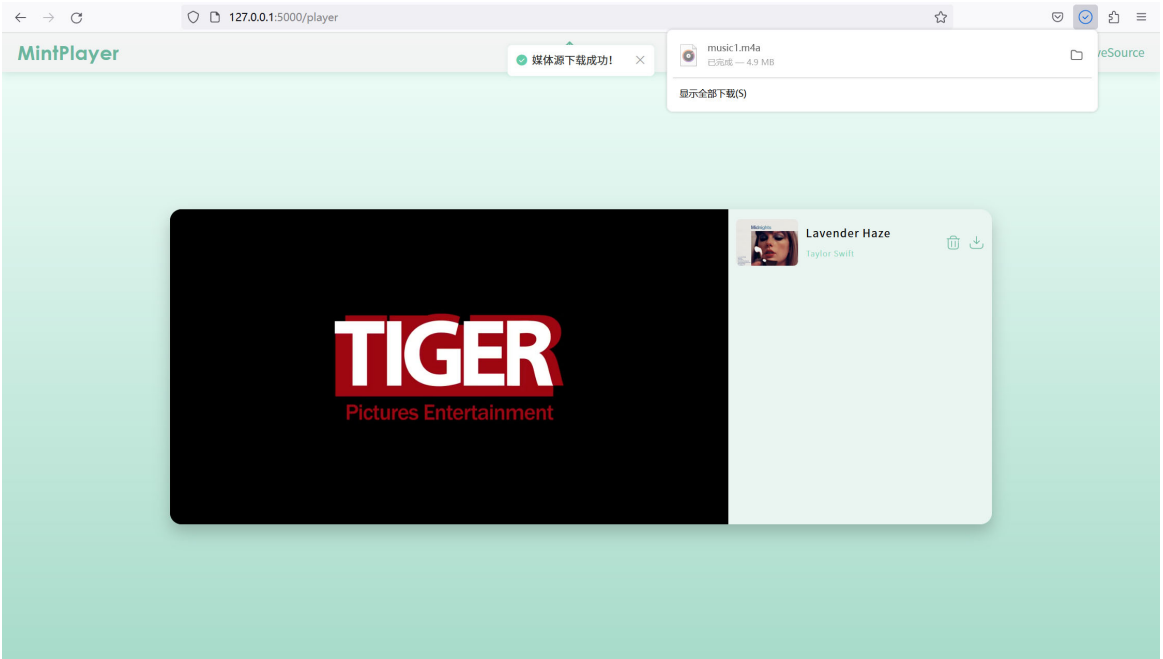


点击播放列表中某一项的删除按钮后，将会发送到后端的 /delete/<int:media_id> 路由进行处理。该路由被@login_required 与我自定义的@admin_required 两个函数修饰器修饰，若用户不具备管理员权限，后端将会返回错误提示的 JSON 与 HTTP 状态码 403，对应权限不足。前端将会弹出上图所示的“您需要管理员权限以进行此操作!”消息提醒，提示用户权限不足。



若用户具备删除操作所需要的管理员操作，则后端会在数据库中查询相应的媒体信息。若找到相应媒体信息，则在数据库中进行删除，并返回新的媒体列表；否则，将会返回错误的 HTTP 状态码 404。如上图所示，“WaveSource”用户具备管理员权限，因此列表中的“流浪地球 2：先导预告片”成功被删除，同时页面弹出“视频删除成功”的消息提示（如下图所示）。

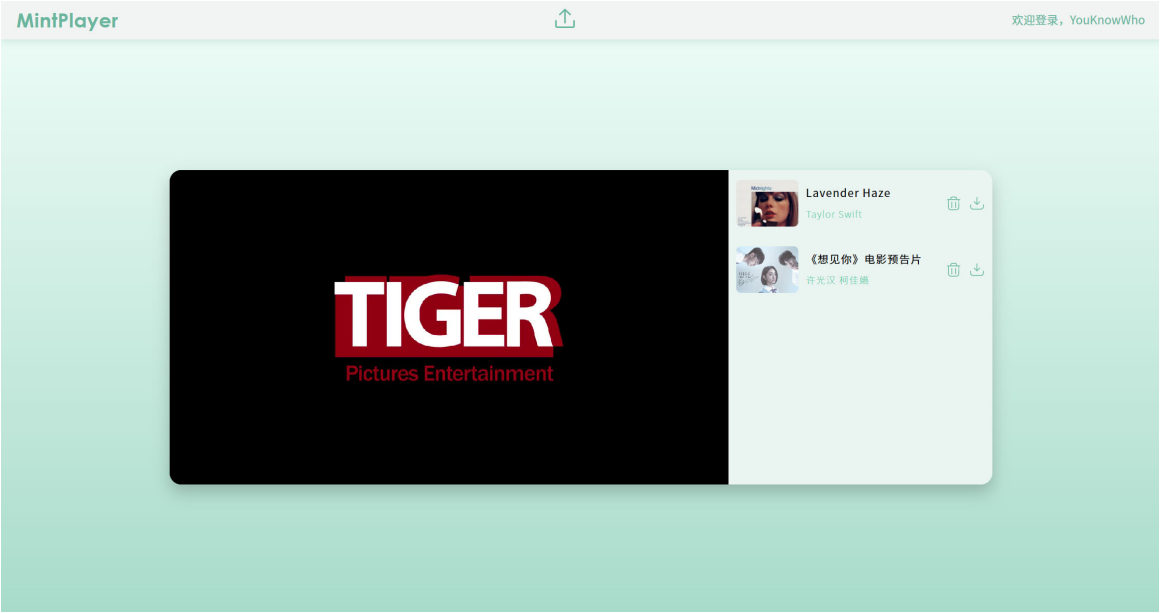




点击播放列表中某一项的下载按钮后，将会发送到后端的 /download/<int:media_id> 路由进行处理。该路由被@login_required 与我自定义的@admin_required 两个函数修饰器修饰，若用户不具备管理员权限，后端将会返回错误提示的 JSON 与 HTTP 状态码 403，对应权限不足。前端将会弹出上图所示的“您需要管理员权限以进行此操作！”消息提醒，提示用户权限不足。若用户具备下载操作所需要的管理员操作，则后端会在数据库中查询相应的媒体信息。若找到相应媒体信息，则通过 send_from_directory 发送；否则，将会返回错误的 HTTP 状态码 404。如上图所示，“WaveSource”用户具备管理员权限，因此列表中的“Lavender Haze”下载到本地成功，并给出消息提示信息“媒体源下载成功！”。

如上图所示，展示了上传视频的页面，对应 url 为 http://127.0.0.1:5000/upload。该页面用于上传视频，并提供了相应的表单和布局。在页面中间的表单中，用户需要填写标题、副标题以及上传媒体缩略图和媒体源（视频）。用户可以在“标题”和“副标题”的输入框中输入视频相关信息；在“媒体缩略图”的输入框中，用户可以选择上传图像文件（仅支持 PNG 和 JPEG 格式）作为视频的缩略图。在“媒体源”的输入框中，用户可以选择上传视频或音频文件。点击上传按钮后，表单的提交会发送到后端的 /upload 路由进行处理。如果用户未填写必要的信息或上传过程中出现错误，后端会将相应的错误消息通过 Flask 的 flash 函数传递到前端，并

在页面中显示提示信息。在页面右侧，有一个用于返回的按钮。用户可以点击该按钮，通过点击 <a> 标签中的链接返回到播放器页面。



根据上图所示，在填写了标题、副标题、缩略图、媒体源等信息后，视频成功被上传到后端并记录在数据库中。媒体播放列表更新了相应的条目。

3.2 实验源代码（部分）

1. 后端代码（application.py）

```
import os
from flask import Flask, render_template, request, flash, url_for, redir
ect, jsonify, send_from_directory
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, login_requi
red, current_user
from werkzeug.utils import secure_filename
from functools import wraps

db = SQLAlchemy()
app = Flask(__name__, static_url_path='')
app.secret_key = '#iLoveWebDesigning20230501'
# 数据库
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///database.db"
# 文件存储路径
app.config['UPLOAD_PIC_FOLDER'] = ".\static\pic"
app.config['UPLOAD_MEDIA_FOLDER'] = ".\static\media"

# 装饰器 要求用户的管理员身份
def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        user = Users.query.filter_by(username = current_user.id).first()
        if not(user.admin):
            return jsonify({'error' : '您需要管理员权限以进行此操作！'}), 4
03
        return f(*args, **kwargs)
    return decorated

db.init_app(app)
# 数据库中存储的用户信息格式
class Users(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String, nullable=False)
    password = db.Column(db.String, nullable=False)
    email = db.Column(db.String, nullable=False)
    admin = db.Column(db.Boolean, nullable=False)

    def __repr__(self):
        return '<User %r>' % self.user
```

```
# 数据库中存储的媒体信息格式
class Media(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String, nullable=False)
    subtitle = db.Column(db.String, nullable=False)
    picSrc = db.Column(db.String, nullable=False)
    mediaSrc = db.Column(db.String, nullable=False)

    def __repr__(self):
        return '<Media %r>' % self.title

    def to_json(self):
        from sqlalchemy.orm import class_mapper
        columns = [c.key for c in class_mapper(self.__class__).columns]
        return dict((c, getattr(self, c)) for c in columns)

login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message = '登录后方可使用 MintPlayer! '
login_manager.login_message_category = "info"

class User(UserMixin):
    pass

# 如果用户名存在则构建一个新的用户类对象，并使用用户名作为 ID
# 如果不存在，返回 None
@login_manager.user_loader
def load_user(_username):
    user = Users.query.filter_by(username = _username).first()
    if user is not None:
        curU = User()
        curU.id = user.username
        return curU

@login_manager.request_loader
def request_loader(request):
    if(request.form.get('username') == None):
        return
    user = Users.query.filter_by(username = request.form['username']).fi
rst()
    if user is not None:
        curU = User()
        curU.id = user.username
        return curU
```

装 订 线

```
@app.route("/")
def home_page():
    return render_template("home.html")


@app.route("/login", methods=["POST", "GET"])
def login():
    if(request.method == "GET"):
        return render_template("login.html")
    else:
        user = Users.query.filter_by(username = request.form['username
']).first()
        if(user and user.password == request.form['password']):
            curr_user = User()
            curr_user.id = request.form['username']

            login_user(curr_user)
            return redirect(url_for('player'))
        else:
            flash('用户名或密码错误！')
            return redirect(url_for('login'))


@app.route("/signup", methods=['POST'])
def signup():
    print(request.form)
    if('username' not in request.form or 'password' not in request.form
or 'email' not in request.form):
        flash('个人信息填写不全！')
        return redirect(url_for('login'))
    user = Users.query.filter_by(username = request.form['username']).fi
rst()
    if user != None:
        flash('该用户名已被占用！')
        return redirect(url_for('login'))
    user = Users(username = request.form['username'], password = reques
t.form['password'], email = request.form['email'], admin=False)
    db.session.add(user)
    db.session.commit()
    # flask log-in 进行登录
    curr_user = User()
    curr_user.id = request.form['username']
    login_user(curr_user)
    return redirect(url_for('player'))
@app.route("/player")
```

装 订 线

```
@login_required
def player():
    return render_template("player.html", name = current_user.id)


@app.route("/player/getMediaList")
@login_required
def getMediaList():
    medias = Media.query.all()
    mediaList = [media.to_json() for media in medias]
    return jsonify({"mediaList": mediaList})


@app.route("/upload", methods=["POST", "GET"])
@login_required
def upload():
    if(request.method == "GET"):
        return render_template("upload.html")
    else:
        files = request.files
        if 'picSrc' not in files or files['picSrc'] == '':
            flash('未上传缩略图！')
            return redirect(request.url)
        elif 'mediaSrc' not in files or files['mediaSrc'] == '':
            flash('未上传媒体源！')
            return redirect(request.url)
        picFileName = secure_filename(files['picSrc'].filename)
        mediaFileName = secure_filename(files['mediaSrc'].filename)
        # 将文件存入到本地
        files['picSrc'].save(os.path.join(app.config['UPLOAD_PIC_FOLDER
'], picFileName))
        files['mediaSrc'].save(os.path.join(app.config['UPLOAD_MEDIA_FO
LDER'], mediaFileName))
        db.session.add(Media(
                            title = request.form['title'],
                            subtitle = request.form['subtitle'],
                            picSrc =  '/pic/' + picFileName,
                            mediaSrc = '/media/' + mediaFileName))
        db.session.commit()
        return redirect(url_for('player'))


@app.route("/download/<int:media_id>", methods=["GET"])
@login_required
@admin_required
```

```python
def download(media_id):
    media = Media.query.filter_by(id = media_id).first()
    if(media == None):
        return jsonify({"error" : "该文件不存在！"}), 404
    mediaName = media.mediaSrc.split("/")[-1]   # 取文件名
    try:
        dir = app.config["UPLOAD_MEDIA_FOLDER"]
        return send_from_directory(dir, mediaName, as_attachment=True)
    # 不存在
    except:
        return jsonify({"error" : "该文件不存在！"}), 404


@app.route("/delete/<int:media_id>", methods=["POST"])
@login_required
@admin_required
def delete(media_id):
    media = Media.query.filter_by(id = media_id).first()
    if(media == None):
        return jsonify({"error" : "该文件不存在！"}), 404
    db.session.delete(media)
    db.session.commit()
    return jsonify({}), 200
```

2. 登录页面前端代码（login.js）

```javascript
const container = document.querySelector('#container');
const signInSwitchButton = document.querySelector('#signIn');
const signUpSwitchButton = document.querySelector('#signUp');
const signInSubmitButton = document.getElementById('signInForm');
const signInPwd = document.getElementById('signInPwd');
const signInPwdMd5 = document.getElementById('signInPwdMd5');
const signUpPwd = document.getElementById('signUpPwd');
const signUpPwdMd5 = document.getElementById('signUpPwdMd5');
signUpSwitchButton.addEventListener('click', () =>
    container.classList.add('right-panel-active')
);

signInSwitchButton.addEventListener('click', () =>
    container.classList.remove('right-panel-active')
);

function signInEncryption(){
```

```
    signInPwdMd5.value = md5(signInPwd.value);
}

function signUpEncryption(){
    signUpPwdMd5.value = md5(signUpPwd.value);
}
```

3. 消息弹窗页面前端代码（message.js）

```
class Message {

    // 构造函数
    constructor() {
        const containerId = 'message-container';
        this.containerEl = document.getElementById(containerId);

        if (!this.containerEl) {
            // 若不存在则创建新的
            this.containerEl = document.createElement('div');
            this.containerEl.id = containerId;
            // 插入 body 的末尾
            document.body.appendChild(this.containerEl);
        }
    }

    // 消息弹窗
    show({ type, text, duration, closeable }) {
        let messageEl = document.createElement('div');
        // 开始显示
        messageEl.className = 'message move-in';
        messageEl.innerHTML = `
            <span class="icon icon-${type}"></span>
            <div class="text">${text}</div>
        `;
        // 若当前按钮可关闭
        if (closeable) {
            // 创建关闭按钮
            let closeEl = document.createElement('div');
            closeEl.className = 'close icon icon-close';
            messageEl.appendChild(closeEl);
            // 监听按钮点击
            closeEl.addEventListener('click', () => {
                this.close(messageEl);
```

```
            });
        }

        // 追加到 message-container 末尾
        this.containerEl.appendChild(messageEl);

        // 计时结束后关闭弹窗
        if (duration > 0) {
            setTimeout(() => {
                this.close(messageEl);
            }, duration);
        }
    }

    // 弹窗关闭
    close(messageEl) {
        // 移除 move-in 效果
        messageEl.className = messageEl.className.replace('move-in', '
');
        // 添加 move-out 效果
        messageEl.className += 'move-out';

        messageEl.addEventListener('animationend', () => {
            messageEl.setAttribute('style', 'height: 0; margin: 0');
        });

        // 动画结束后删除
        messageEl.addEventListener('transitionend', () => {
            messageEl.remove();
        });
    }
}
```

装 订 线

4. 播放器前端代码（player.js）

```
const video = document.getElementById("video")
const videoName = document.getElementById("video-name")
const videoContainer = document.getElementById("video-container")
const videoControls = document.getElementById("video-control-bar")

const playpause = document.getElementById("playpause");
const mute = document.getElementById("mute");
const picInPic = document.getElementById("picinpic");
```

```
const fullscreen = document.getElementById("fs");
const currentTime = document.getElementById("current-time");
const totalTime = document.getElementById("total-time");

const progress = document.getElementById("progress");
const progressDot = document.getElementById("progress-dot");
const progressBar = document.getElementById("progress-bar");

const volumeValue = document.getElementById("volume-value");
const volumeBar = document.getElementById("volume-bar");
const volumeBarDone = document.getElementById("volume-bar-done");

const multipleSpeedList = document.getElementById("multiple-speed-choic
es").getElementsByTagName('li');

const videoAlternativeBar = document.getElementById("video-chooser")

var mouseDragForProgress = false;          // 鼠标拖拽进度条
var mouseDragForVolume = false;            // 鼠标拖拽音量条

video.controls = false;          // 禁用自带的 controls

// 视频选择列表
let focusAlterOrder = 0;
var videoAlternativeList = [];


// 向后端发送 GET 请求获取视频播放列表
function getMediaList(){
    var xhr = new XMLHttpRequest();
    xhr.responseType = 'json';
    xhr.open("get", "/player/getMediaList");
    xhr.send();
    xhr.onreadystatechange = ()=>{
        if(xhr.readyState == xhr.DONE && xhr.status == 200){
            videoAlternativeList = xhr.response['mediaList'];
            loadVideoAlter();
        }
    }
    xhr.onerror = (error)=>{
        console.log(error);
    }
}
getMediaList();
```

```
// 填充视频列表某元素的 Html 属性
function fillVideoAlter(item, order){
    const title = "<p class = \"title\">" + item.title + "</p>";
    const subtitle = "<p class = \"subtitle\">" + item.subtitle + "</p>
";
    const detail = "<div class = \"detail\" id = \"videoAlter"+ String(o
rder) + "\">" + title + subtitle + "</div>";
    const videopic = "<img class = \"videosrc\" src=\"" + item.picSrc +"
\">";
    const deletebutton = "<img id = \"deleteButton" + String(order) + "\
" src=\"./icon/delete.svg\" class=\"button\">";
    const downloadbutton = "<img id = \"downloadButton" + String(order)
+ "\" src=\"./icon/download.svg\" class=\"button\">";
    videoAlternativeBar.innerHTML += "<div class = \"video-alternative\
">" + videopic + detail + deletebutton + downloadbutton + "</div>";
}


// 改变视频列表顺序
function changeVideoAlter(){
    videoAlternativeBar.innerHTML = "";
    // 当前选中的视频在列表首位
    fillVideoAlter(videoAlternativeList[focusAlterOrder], focusAlterOrd
er);
    for(const order in videoAlternativeList){
        if(videoAlternativeList[order] != videoAlternativeList[focusAlte
rOrder])
            fillVideoAlter(videoAlternativeList[order], order);
    }

}


// 加载视频选择列表
function loadVideoAlter(){
    changeVideoAlter();
    for(let order in videoAlternativeList){
        document.getElementById("deleteButton"+ String(order)).addEvent
Listener("click", (e) => {
            var xhr = new XMLHttpRequest();
            xhr.responseType = 'json';
            xhr.open("post", "/delete/" + String(videoAlternativeList[or
der].id));
            xhr.send();
            xhr.onreadystatechange = ()=>{
```

```
                if(xhr.readyState == xhr.DONE && xhr.status == 200){
                    console.log('test')
                    // 弹窗提示
                    const message = new Message();
                    message.show({
                        type: 'success',
                        text: '视频删除成功！',
                        duration: 2000,
                        closeable: true,
                    });
                    getMediaList();
                }
                else{
                    const message = new Message();
                    message.show({
                        type: 'warning',
                        text: xhr.response['error'],
                        duration: 2000,
                        closeable: true,
                    });
                }
            }
            xhr.onerror = (error)=>{
                console.log(error);
            }
        });

        document.getElementById("downloadButton"+ String(order)).addEve
ntListener("click", (e) => {
            var xhr = new XMLHttpRequest();
            xhr.responseType = 'blob';
            xhr.open("get", "/download/" + String(videoAlternativeList[o
rder].id));
            xhr.send();
            xhr.onreadystatechange = ()=>{
                if(xhr.readyState == xhr.DONE && xhr.status == 200){
                    var blob = xhr.response;
                    var link = document.createElement('a');
                    link.href = window.URL.createObjectURL(blob);
                    link.download = xhr.getResponseHeader('Content-Dispo
sition').split("filename=")[1];

                    // 隐藏下载链接
                    link.style.display = 'none';
```

```
                // 将链接添加到文档中
                document.body.appendChild(link);

                // 触发下载
                link.click();

                // 清理资源
                document.body.removeChild(link);
                window.URL.revokeObjectURL(link.href);

                // 弹窗提示
                const message = new Message();
                message.show({
                    type: 'success',
                    text: '媒体源下载成功！',
                    duration: 2000,
                    closeable: true,
                });
            }
            else{
                // 响应是 JSON 数据
                var reader = new FileReader();

                reader.onload = () => {
                    var responseData = JSON.parse(reader.result);

                    // 弹窗提示
                    const message = new Message();
                    message.show({
                        type: 'warning',
                        text: responseData['error'],
                        duration: 3000,
                        closeable: true,
                    });
                };

                reader.readAsText(xhr.response, 'utf-8');
            }
        }
        xhr.onerror = (error)=>{
            console.log(error);
        }
    });

    document.getElementById("videoAlter"+ String(order)).addEventLi
```

```
stener("click", (e) => {
            video.src = videoAlternativeList[order]['mediaSrc'];
            videoName.innerHTML = videoAlternativeList[order]['title'];
            focusAlterOrder = order;
            changeVideoAlter();
            totalTime.innerHTML = getFormatTime(video.duration, video.du
ration);    // 初始化视频长度
            progressDot.style.marginLeft = 0;
                                        // 初始化圆点位置
            // 音频用封面图
            if(videoAlternativeList[order]['mediaSrc'].slice(-4) == ".m4
a")
                document.getElementById("video-container").style.backgro
undImage = "url(" + videoAlternativeList[order]['pic'] + ")";
            // 其他的为黑色底色(background 设置为 None 的话会影响其他 css 属性,
再次设置图片要重设，因此此处也用 Image)
            else
                document.getElementById("video-container").style.backgro
undImage = "linear-gradient(to bottom, black, black)";
            loadVideoAlter();
        });
    }
}

// 释放拖动进度条
document.body.addEventListener("mouseup", (e) => {
    mouseDragForProgress = false;
    mouseDragForVolume = false;
});


// 监听鼠标移动
document.body.addEventListener("mousemove", (e) => {
    if(mouseDragForProgress){
        const barRec = progress.getBoundingClientRect();
        const pos = (e.pageX - barRec.left) / progress.offsetWidth;
        video.currentTime = pos * video.duration;
    }
    else if(mouseDragForVolume){
        const barRec = volumeBar.getBoundingClientRect();
        var pos = (barRec.bottom - e.pageY) / volumeBar.offsetHeight;
        if(pos > 1)
            pos = 1;
        else if(pos < 0)
            pos = 0;
```

```
        video.volume = pos;
    }
});

// 监听全屏
window.addEventListener('fullscreenchange', () => {
    const isFullScreen = document.fullScreen || document.mozFullScreen |
| document.webkitIsFullScreen
    if(isFullScreen) {
        fullscreen.src = "./icon/fullscreenexit.svg";
    }
    else{
        fullscreen.src = "./icon/fullscreen.svg";
    }

    // 防止进度条圆点错位
    progressDot.style.marginLeft = (progress.value / video.duration) * p
rogress.offsetWidth - progressDot.offsetWidth / 2 + "px";
})


// 播放按钮
playpause.addEventListener("click", (e) => {
    if (video.paused || video.ended) {
        video.play();
        playpause.src = "./icon/pause.svg"
    }
    else {
        video.pause();
        playpause.src = "./icon/play.svg"
    }
});


// 点击视频本身同理暂停/播放
video.addEventListener("click", (e) => {
    if (video.paused || video.ended) {
        video.play();
        playpause.src = "./icon/pause.svg"
    }
    else {
        video.pause();
        playpause.src = "./icon/play.svg"
    }
});
```

装 订 线

```
// 监听音量变化
video.addEventListener("volumechange", (e)=>{
    volumeValue.innerHTML = Math.floor(video.volume * 100);
    volumeBarDone.style.height = video.volume * volumeBar.offsetHeight +
 'px';
});


// 点击音量条
volumeBar.addEventListener("click", (e) => {
    const barRec = volumeBar.getBoundingClientRect();
    var pos = (barRec.bottom - e.pageY) / volumeBar.offsetHeight;
    if(pos > 1)
        pos = 1;
    else if(pos < 0)
        pos = 0;
    video.volume = pos;
});

// 拖动音量条
volumeBar.addEventListener("mousedown", (e) => {
    mouseDragForVolume = true;
})


// 静音键
mute.addEventListener("click", (e) => {
    video.muted = !video.muted;
    if(video.muted)
        mute.src = "./icon/mute.svg";
    else
        mute.src = "./icon/volume.svg";
});

// 画中画
picInPic.addEventListener("click", () => {
    video.requestPictureInPicture();
})


// 同步进度条时间
video.addEventListener("timeupdate", () => {
    progress.setAttribute("max", video.duration);
```

装 订 线

```
    progress.value = video.currentTime;

    // 控制圆点
    progressDot.style.marginLeft = (progress.value / video.duration) * p
rogress.offsetWidth - progressDot.offsetWidth / 2 + "px";

    // 时间显示
    currentTime.innerHTML = getFormatTime(video.currentTime, video.dura
tion);
    totalTime.innerHTML = getFormatTime(video.duration, video.duratio
n);
});


// 点击进度条
progress.addEventListener("click", (e) => {
    const barRec = progress.getBoundingClientRect();
    const pos = (e.pageX - barRec.left) / progress.offsetWidth;
    video.currentTime = pos * video.duration;
});


// 拖动进度条
progress.addEventListener("mousedown", (e) => {
    mouseDragForProgress = true;
});


// 全屏
fullscreen.addEventListener("click", (e) => {
    if (document.fullscreenElement !== null) {
        // 当下处于全屏模式
        document.exitFullscreen();
    }
    else {
        // 当下并非全屏模式
        videoContainer.requestFullscreen();
    }
    videoContainer.setAttribute("data-fullscreen", !!document.fullscree
nElement);
});


// 根据总时长格式化为 00:00 或 00:00:00 的格式
function getFormatTime(time, duration) {
```

```
    var time = time || 0;

    var h = parseInt(time / 3600),
        m = parseInt(time % 3600 / 60),
        s = parseInt(time % 60);

    s = s < 10 ? "0" + s : s;

    if (duration >= 3600){
        m = m < 10 ? "0" + m : m;
        h = h < 10 ? "0" + h : h;
        return h + ":" + m + ":" + s;
    }
    else{
        if(m == 0)
            m = "00";
        else
            m = m < 10 ? "0" + m : m;
        return m + ":" + s;
    }
}

// 监听倍速列表中的元素
var speedSelected = multipleSpeedList[3];
speedSelected.style.color = "mediumaquamarine";
for(var i = 0; i < multipleSpeedList.length; i++){
    multipleSpeedList[i].addEventListener('click', function() {
        video.playbackRate = parseFloat(this.innerHTML);
        speedSelected.style.color = "white";
        speedSelected = this;
        speedSelected.style.color = "mediumaquamarine";
    })
}


// 开启视频循环
function openVideoCycle(obj){
    if(obj.checked){
        video.loop = true;
    }
    else
        video.loop = false;
}

// 开启视频镜像
```

```
function openVideoMirror(obj){
    if(obj.checked){
        video.style.transform = "rotateY(180deg)";
    }
    else
        video.style.transform = "none";
}
```

5. 登录注册页面 HTML（login.html）

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale
=1.0">
        <title>MintPlayer - 登录注册</title>
        <link rel="stylesheet" href="/css/login.css">
    </head>

    <body>
        <div class='container' id='container'>
            <div class="form-container sign-up-container">
            <form id="signUpForm" action="/signup" method="post" onsubmi
t="signUpEncryption()">
                <h1>用户注册</h1>
                <input type="text" name="username" placeholder="用户名" r
equired>
                <input type="password" id="signUpPwd" placeholder="密码"
 required>
                <input type="hidden" id="signUpPwdMd5" name="password">
                <input type="email" name="email" placeholder="邮箱" requ
ired>
                {% with messages = get_flashed_messages() %}
                    <div class="hint">{{ messages[0] }}</div>
                {% endwith %}
                <button type="submit">注册</button>
            </form>
            </div>
            <div id="signInForm" class="form-container sign-in-container
">
                <form action="/login" method="post" onsubmit="signInEncr
yption()">
```

```
                    <h1>用户登录</h1>
                    <input type="text" name="username" placeholder="用户
名" required>
                    <input type="password" id="signInPwd" placeholder="
密码" required>
                    <input type="hidden" id="signInPwdMd5" name="passwor
d">
                    {% with messages = get_flashed_messages() %}
                        <div class="hint">{{ messages[0] }}</div>
                    {% endwith %}
                    <button type="submit">登录</button>
                </form>
            </div>

            <div class="overlay-container">
                <div class="overlay">
                    <div class="overlay-panel overlay-left">
                        <h1>已有帐号？</h1>
                        <p>点击下方按钮登录 MintPlayer</p>
                        <button class='switch' id="signIn">登录</button>
                    </div>
                    <div class="overlay-panel overlay-right">
                        <h1>还未注册？</h1>
                        <p>点击注册你的 MintPlayer 账号</p>
                        <button class='switch' id="signUp">注册</button>
                    </div>
                </div>
            </div>
        </div>
    </body>
    <script src="/js/login.js"></script>
    <script src="/js/md5.js"></script>
</html>
```

装 订 线

6. 上传页面 HTML（upload.html）

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale
=1.0">
        <title>MintPlayer - 视频上传</title>
```

```
        <link rel="stylesheet" href="/css/upload.css">
    </head>

    <body>
        <div class='container' id='container'>
            <div id="signInForm" class="form-container sign-in-container
">
                <form action="/upload" method="post" enctype="multipart/
form-data" onsubmit="encryption()">
                    <label>标题</label>
                    <input type="text" name="title" placeholder="在此输入
标题" required>
                    <label>副标题</label>
                    <input type="text" name="subtitle" placeholder="在此
输入副标题" required>
                    <label>媒体缩略图</label>
                    <input type="file" accept="image/png, image/jpeg" na
me="picSrc" required>
                    <label>媒体源</label>
                    <input type="file" accept="video/mp4" name="mediaSrc
" required>
                    {% with messages = get_flashed_messages() %}
                        <div class="hint">{{ messages[0] }}</div>
                    {% endwith %}
                    <button type="submit">上传</button>
                </form>
            </div>

            <div class="overlay-container">
                <div class="overlay">
                    <div class="overlay-panel overlay-right">
                        <h1>上传视频</h1>
                        <p>为 MintPlayer 提供优质资源</p>
                        <a href="{{ url_for('player') }}">
                            <button class='switch' id="signUp">返回</butt
on>
                        </a>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

装 订 线

四、心得体会

在本次的服务端编程实验中，我在实验三的基础上成功地实现了一个具有登录注册功能以及用户身份分级的的媒体播放器。通过学习并运用服务端编程的 Flask 以及其拓展的 Flask-SQLAlchemy 和 Flask-Login 等后端技术，实现了前后端交互、表单信息传输和视频下载等功能。

通过使用 Flask 框架，我能够更加灵活地处理用户的登录和注册请求，并将用户信息存储到数据库中。借助 Flask-SQLAlchemy 扩展，我得以更方便地与数据库进行交互，方便快捷地实现用户信息的存储和读取以及媒体播放列表的修改与读取。

在本次实验中，我学习了用户身份认证相关的知识，并在媒体播放器应用中应用了 Flask-Login 提供的@login_required 装饰器和我自定义的@admin_required 装饰器。通过使用@login_required 装饰器，我能够保护网站中需要登录才能访问的页面和功能（如播放器页面和一些相应的 api）。在需要进行身份认证的视图函数上添加@login_required 装饰器后，如果用户未登录，Flask-Login 会自动将其重定向到登录页面。这样可以确保只有经过身份认证的用户才能访问受保护的资源，增加了应用的安全性。除了登录认证，我还自定义了@admin_required 装饰器，以限制只有管理员才能进行的某些操作。通过在视图函数上添加@admin_required 装饰器，我可以确保只有具有管理员权限的用户才能执行敏感的操作，如下载和删除视频等，从而有效地保护应用中的重要功能，防止非授权用户进行误操作或滥用权限。在实现@admin_required 装饰器时，我结合了 Flask-Login 提供的当前用户信息获取功能，通过判断当前用户是否为管理员从而动态地进行权限控制。

此外，本次的视频播放器还具备视频上传的功能。通过前端表单，用户可以输入视频的标题、副标题，并选择相应的缩略图和媒体源进行上传。后端代码接收到表单数据后，将文件保存到指定的文件夹，并将相关信息存储到数据库中。对应的，我也实现了管理员的视频下载功能，能够将后端传输的音视频文件下载到用户本地。具有管理员权限的用户还能够修改媒体播放列表，删除不希望出现的相应媒体，后端将会在验证管理员身份之后在数据库中进行相应信息的删改。

通过这次实验，我不仅加深了对前端开发中 JavaScript 与后端交互的理解，还

学习了后端开发中 Flask 框架的使用和相关扩展的应用。这对我今后从事网页开发和应用程序设计都具有重要意义。我将继续努力，将所学到的知识应用到实践中，不断提升自己的技术水平和网页设计能力。