同济大学

TONGJI UNIVERSITY

《WEB 技术》

实验报告

实验名称	服务端编程
小组成员	
学院（系）	电子与信息工程学院
专　　业	计算机科学与技术专业
任课教师	郭玉臣
日　　期	2023 年　5 月 22 日

1. 实验原理

本实验涉及的技术主要包括服务端开发、数据库使用和前端开发技术。通过服务端开发，可以在服务器端实现一个后端服务，接收来自客户端的请求，响应并处理这些请求。通过使用 MySQL 数据库存储和管理数据，并在 Django 后端服务中实现数据的读取、修改、更新或删除等操作。前端开发则负责将从后端获取到的数据显示在网页中，同时也提供了用户和管理员交互的界面。本实验将建立在实验 3 的基础上，为视频播放添加管理页面提供后端服务。

2. 实验基本要求

基本功能：

2.1 用户注册登录模块：实现账号注册、登录操作，通过数据库保存用户信息并验证登录状态。其中可以对密码和用户名进行正则表达式验证、敏感字监测、用户名重复等基本注册操作。进行设置 2 个角色，一个游客，一个管理员。游客只能播放视频，管理员可以修改播放列表。

2.2 后端获取播放列表并进行播放：从后端数据库获取播放列表，实现视频播放操作。

2.3 管理员可编辑列表：管理员可以对播放列表进行增删改查等操作，并将结果返回到后端处理，数据库中进行相应的增删改查操作。删除操作可以进行多重选择，进行批量删除。

2.4 视频内容预先存储：预先存储视频在本地，模拟存储在服务器内部的静态资源文件，后端数据库返回文件关键字，前端可调取播放。

扩展功能：

2.5 可在线上传视频内容：提供了视频上传功能，方便管理员添加新的视频内容。管理员可以将视频在线上传到管理员列表，后端进行对应操作。

2.6 管理员可以下载视频到本地：管理员可以通过后端服务从服务器下载视频到本地，下载视频指定路径并正常播放

3. 实验内容

小节目录：

装 订 线

3.1 后端框架结构与设置

本实验后端框架选择 Python 语言构建的 Django 框架，该框架可以满足：

1. 快速开发：提供高度集成开发环境，利用 Python 语言的简洁性优势高效开发
2. 内置安全特性：其中在框架主设置中可以自动设置跨域站点请求(CSRF)、脚本注入(XSS)等安全处理，在该项目中也遇到关于跨域站点请求的访问问题，在本地测试时需要特别注意。
3. 其对象关系映射容易上手，通过迁移指令完成 MySQL 数据库的映射，在 models.py 中通过对数据库表建立类的方式轻松操作数据库内容。其提供的 API 更易于理解

3.1.1 主框架 settings

后端设置首先需要在主框架设置中添加网页应用、设置安全策略、设置数据库路径、设置根目录索引等。

设置网页应用：

```
INSTALLED_APPS = [
    "simpleui",
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    'videoplayer',
]
```

查看 Django Web 中间件（安全策略等）

```
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

设置网页模板页面的路径地址：

```
TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [os.path.join(BASE_DIR, 'templates')],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages"
,
            ],
        },
    },
]
```

装

设置数据库路径、密码、数据库名称、主机名和端口号

```
DATABASES= {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'videoplayer',
        'USER': '****',
        'PASSWORD': '****************',
        'HOST': '127.0.0.1',
        'PORT':  '3306',
    }
}
```

订

设置静态文件存储的根路径，此处主要用于模拟服务器静态文件存储，相关的视频文件以及主网页文件的 CSS,JS 文件存储在此

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = "videoplayer/static/"
```

线

3.1.2 主框架 urls 路由设置

设置基本路由，此处指需要到对应的网页应用(videoplayer)当中的路由配置中进行查找

```
urlpatterns = [
    path("videoplayer/", include('videoplayer.urls')),
]
```

3.1.3 videoplayer 应用 urls 路由设置

设置应用内部路由，其中带有 admin 字段的是管理员登录界面，带有 login 或 register 字段的是游客登录、注册界面，带有 delete、upload、down 字段的是管理员界面对应功能的状态显示界面

```
1.    urlpatterns = [
2.        path('toadmin/', views.toAdmin_view),
3.        path('admin/', views.admin_view),
4.        path('', views.toLogin_view),
5.        path('login/', views.Login_view),
6.        path('main/', views.main_view),
7.        path('toregister/', views.toregister_view),
8.        path('register/', views.register_view),
9.        path('uploaded/', views.upload_video),
10.       path('deleteItems/', views.deleteItems),
11.       path('deleted/', views.deleted),
12.       path('downItems/', views.downItems),
13.   ]
```

3.1.4 videoplayer 视图函数

视图函数的作用是对前端数据进行处理，与后端数据库进行交互，实现前后端连接交流，各函数如下，详细实现在功能部分展开。

```
1.    # Create your views here.
2.
3.    def toAdmin_view(request):
4.    def admin_view(request):
5.    def toLogin_view(request):
6.    def Login_view(request):
7.    def main_view(request):
8.    def toregister_view(request):
9.    def register_view(request):
10.   def upload_video(request):
11.   def deleteItems(request):
12.   def deleted(request):
13.   def downItems(request):
```

3.1.5 Models 数据库表对象

Models 相当于将每一个表单映射成为一个类，通过操作这些类的实例化对象来在 Django 框架中操作数据库内容。中间省略了 MySQL 自带表单

装 订 线

```
1.    class AdminTable(models.Model):
2.        admin_id = models.IntegerField(primary_key=True)
3.        admin_name = models.CharField(max_length=45, blank=True, null=Tru
    e)
4.        admin_psw = models.CharField(max_length=45, blank=True, null=True
    )
5.
6.        class Meta:
7.            managed = False
8.            db_table = 'admin_table'
1.    class VideoName(models.Model):
2.        video_id = models.IntegerField(primary_key=True)
3.        video_name = models.CharField(max_length=45, blank=True, null=Tru
    e)
4.
5.        class Meta:
6.            managed = False
7.            db_table = 'video_name'
8.
9.
10.   class VideoplayerStudentinfo(models.Model):
11.       stu_id = models.CharField(primary_key=True, max_length=20)
12.       stu_name = models.CharField(max_length=20)
13.       stu_psw = models.CharField(max_length=20)
14.
15.       class Meta:
16.           managed = False
17.           db_table = 'videoplayer_studentinfo'
```

其中 AdminTable 代表管理员表单，VideoName 存储播放视频的名称、序号等，VideoplayerStudentInfo 存储普通游客注册信息。

3.2 数据库建表

其本应用中的表单一共有三个，分别为管理员表单，视频名称表单、用户注册表单。

管理员表单：

admin_id	admin_name	admin_psw
1	admin	114514
NULL	NULL	NULL

管理员表单中包括管理员编号、用户名、密码，其中 ID 作为主键。

视频名称表单：

video_id	video_name
1	Under Scope: Liz and Blue Bird Review
2	Song Birds MV1
3	Nozomi Autography
4	An Open letter to Kyoto Animation
5	Song Birds MV2
6	the Third Movement
7	Liz and Blue Bird Preview
8	Liz and Blue Bird Snippet
10	yyyccc

视频名称表单中包括视频的序号以及名称，视频序号作为主键。其中 video_id 记录的时自服务开始以来上传的视频编号，所以会有不连续的情况。由于视频在存储时以顺序编码，所有视频路径名称的处理都会在中台或者前端完成。

用户注册表单：

stu_id	stu_name	stu_psw
00005	zhangsan	123456
138531	beewbe	2323
419256	dan	dandan
NULL	NULL	NULL

用户注册表单中包括用户 ID，用户名、密码，其中用户 ID 作为主键。

3.3 页面路由

http://127.0.0.1:8000/videoplayer/： 游客登录界面，用户输入用户名和密码进行登录，后端进行数据库检索查询，跳转到主界面

http://127.0.0.1:8000/videoplayer/toregister/：用户注册界面，用户输入用户名和密码后进行注册，跳转到主界面

http://127.0.0.1:8000/videoplayer/toadmin/：管理员登录界面，登录成功后跳转到管理员管理界面

http://127.0.0.1:8000/videoplayer/main/ ：视频观看主界面，上一实验结果

http://127.0.0.1:8000/videoplayer/admin/：管理员管理界面，包括视频的增删改查和下载等

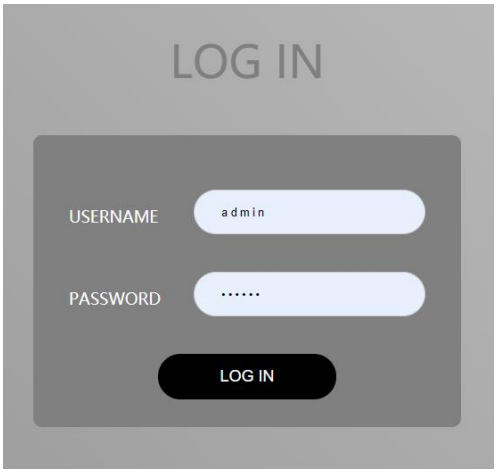
.../register, .../uploaded, .../deleteItems, .../deleted, .../downItems 均为上述页面对应的响应界面

装 订 线

3.4 基本功能实现

3.4.1 游客登录功能

进入游客登录界面(http://127.0.0.1:8000/videoplayer/)，游客选择输入用户名和密码，点击 LOG IN，当用户二者之一没有输入，则提示“请输入正确的账号和密码”。当用户名称和密码不匹配时，提示“账号和密码错误”，用户名和账号匹配时进入主页面。



请输入正确账号和密码!　　账号密码错误!

首先进入到页面在 views.py 中实现页面渲染

```
1.    def toLogin_view(request):
2.        return render(request, 'login.html')
```

1. HTML 部分：

```
1.    <form action="/videoplayer/login/" method="post" class="former">
2.            {% csrf_token %}
3.            <div class="items username">
4.              <label>USERNAME</label> <input name="user" type="text" />
5.            </div>
6.            <div class="items password">
7.              <label>PASSWORD</label> <input name="pwd" type="password" /
    >
8.            </div>
9.            <input type="submit" value="LOG IN" />
10.   </form>
```

HTML 部分利用 form 表单提出跳转，其中定义 post 请求方法、跳转 URL 位置。{% csrf_token %} 为 Django 语法，表示 post csrf 令牌，防止 Django 默认安全机制阻止网站访问，提交部分用 HTML input 标签，将数据作为表单结果提交给目标 URL。经过应用内部路由后，进入 views.py 进行登录判断。

2. views.py 登录判断

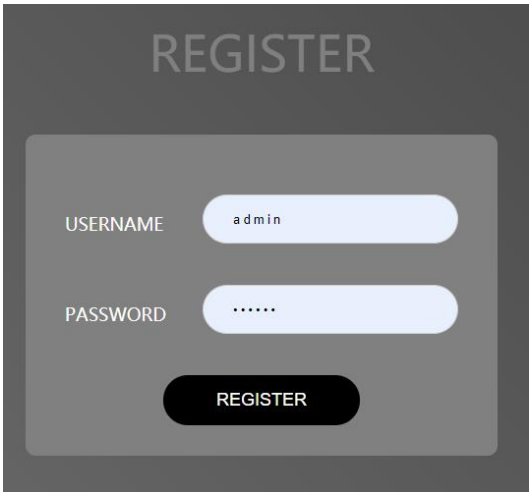
```
1.    def Login_view(request):
2.        u = request.POST.get("user", '')
3.        p = request.POST.get("pwd", '')
4.
5.        videos = VideoName.objects.all()
6.        data = list(videos.values())
7.        rt = {'data': json.dumps(data)}
8.
9.        if u and p:
10.           c = VideoplayerStudentinfo.objects.filter(stu_name=u, stu_psw
    =p).count()
11.           if c >= 1:
12.               return render(request, 'main.html', rt)
13.           else:
14.               return HttpResponse("账号密码错误！")
15.       else:
16.           return HttpResponse("请输入正确账号和密码!")
```

利用 request.POST.get()的方法获取表单中的数据，在下方判断逻辑处进行判断。在进行判断时，通过 VideoplayerStudentInfo.objects.filter().count()可以通过 models 在数据库中对数据进行筛选。、

同时我们需要把现有数据库的视频列表信息从数据库中获取到，通过 rt 变量构建视频列表的 json 信息，传递给 main.html 页面进行渲染。

3.4.2 游客注册功能

进入游客注册界面(http://127.0.0.1:8000/videoplayer/toregister)，游客选择输入用户名和密码，点击 REGISTER，后端会首先进行是否输入的判断，如有字段没有输入则提示“请输入完整的账号和密码”，若输入成功，则会判断输入用户名是否与以往用户名重复、用户密码是否强度满足。否则提示“用户名重复”或者例如“请输入 8 位数以上的密码，至少包含一个数字和大写字母”



首先进入注册页面实现渲染

```
def toregister_view(request):
    return render(request, "register.html")
```

1. HTML 部分

该部分与登录部分的 HTML 类似，均为表单提交跳转

```
<form action="/videoplayer/register/" method="post" class="former">
        {% csrf_token %}
    <div class="items username">
        <label>USERNAME</label> <input name="user" type="text" />
    </div>
      <div class="items password">
       <label>PASSWORD</label> <input name="pwd" type="password" />
    </div>
    <input type="submit" value="REGISTER" />
 </form>
```

2. views.py 注册判断

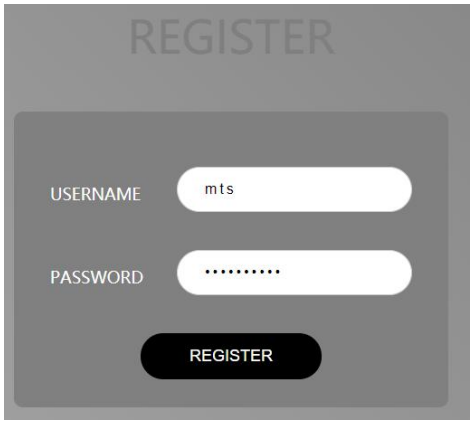
```
def register_view(request):
    u = request.POST.get("user", '')
    p = request.POST.get("pwd", '')

    videos = VideoName.objects.all()
    data = list(videos.values())
    rt = {'data': json.dumps(data)}

```

```
9.        if VideoplayerStudentinfo.objects.filter(stu_name=u).count() != 0
    :
10.           return HttpResponse("用户名已经被占用")
11.       if len(p) < 8:
12.           return HttpResponse("请输入 8 位以上的密码")
13.
14.       flg = 0
15.       for i in range(len(p)):
16.           if p[i].isdigit() or p[i].isupper():
17.               flg = 1
18.       if flg == 0:
19.           return HttpResponse("密码至少包含一个数字和大写字母")
20.
21.       if u and p:
22.           stu_id = random.randrange(111111, 999999)
23.           while VideoplayerStudentinfo.objects.filter(stu_id=stu_id).co
    unt() != 0:
24.               stu_id = random.randrange(111111, 999999)
25.           stu = VideoplayerStudentinfo(stu_id=random.randrange(111111,
    999999), stu_name=u, stu_psw=p)
26.           print(stu)
27.           stu.save()
28.           return render(request, 'main.html', rt)
29.       else:
30.           return HttpResponse("请输入完整的账号和密码")
```

该部分对密码和用户名进行了重复检查、强度检查。通过利用 filter 函数检索用户注册表进行用户名检查。在进行强度检查时，要求用户的密码中必须至少包含一个大写字母、一个数字。同时在写入数据库时为每个用户设置 6 位随机的用户编号作为用户 id 存储在数据库中。同时考虑到该用户 id 作为区分用户的重要标志，在进行用户编号设置时会在原有数据库中进行判重，保证每一个用户的主键编号不同。

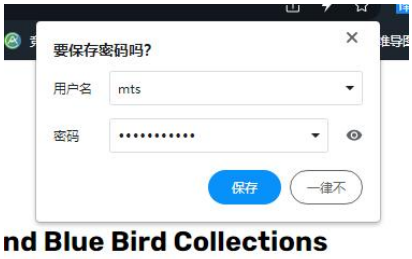
注册效果：

用户注册：输入用户名 mts，密码 Ming112233，点击注册



装 订 线

跳转到主页面显示

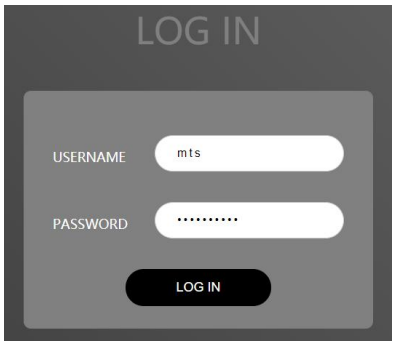


查看数据库并刷新用户注册表单

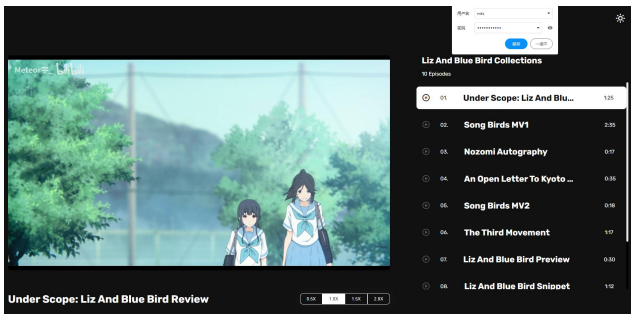
	stu_id	stu_name	stu_psw
	00005	zhangsan	123456
	138531	beewbe	2323
	419256	dan	dandan
▶	910194	mts	Ming112233
*	NULL	NULL	NULL

可以看到系统分配 mts 的 id 为 910194，用户名、密码均正确存储在数据库中。

进入登录页面，输入刚才注册的用户名以及密码

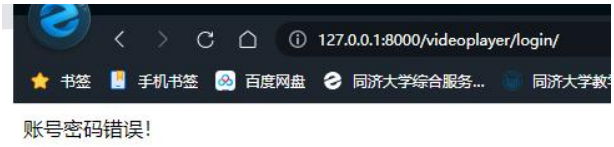


可以观察到正常进入到主页面



当输入密码错误时显示提示

装 订 线

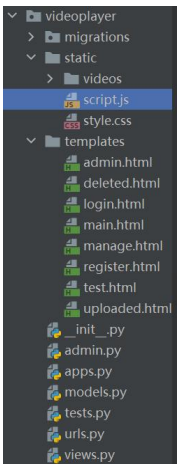


则注册登录模块测试正常，游客可以进行正常注册登录

3.4.3 主界面显示功能

主界面在上一项目中已经构建，本项目主要需要通过 Django 框架获取到后端数据库中的播放列表，利用 JS 对获取到的影片名称进行处理，计算影片地址并加载。

本项目中，主界面的静态文件同一储存在该 videoplayer 应用的 static 文件夹下，通过修改主 settings 中静态目录的根目录引导 html 寻找并加载静态文件。



在主设置中：

```
1. STATIC_URL = "videoplayer/static/"
```

在主页面外联文件中使用 Django 语法获取静态文件路径

```
1.    {% load static %}
2.    <link rel="stylesheet" href="{% static 'style.css' %}" />
1.    {% load static %}
2.    <script src="{% static 'script.js' %}"></script>
```

首先我们需要获取到 videos 静态文件的目录，由于在 Django 中脱离 template 文件无法用 Django 语法获取到静态文件路径，所以需要在原有的 template 中获取到静态文件路径

```
1.    <script>
2.          var videoURL = "{% static 'videos/video' %}"
```

```
        var context = '{{ data }}';
        console.log(context)
    </script>
```

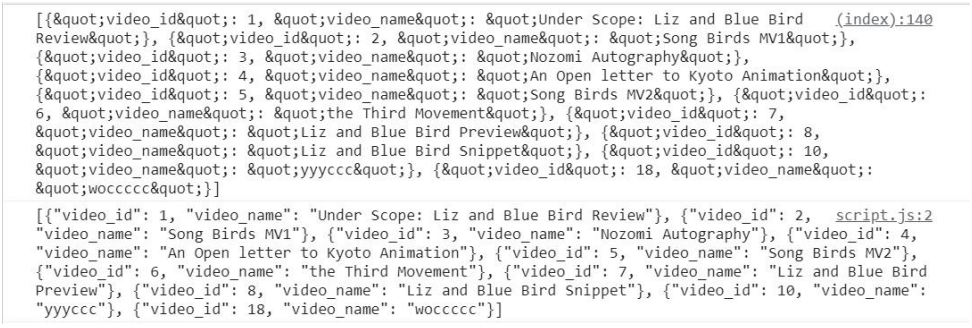
其中 videoURL 伪静态文件所在目录，context 为 videoName 对应的数据库表单信息。

在 JS 中我们首先加载 JSON 文件，获取播放列表；向播放列表中动态添加元素即可

首先解析 JSON 文件

```
    cleanedText = context.replaceAll('&quot;', '"');
    console.log(cleanedText)
    cv_list = JSON.parse(cleanedText)
    console.log(cv_list);
```

观察到 JSON 文件中有 HTML 特殊字符 &qout;，需要手动进行清除



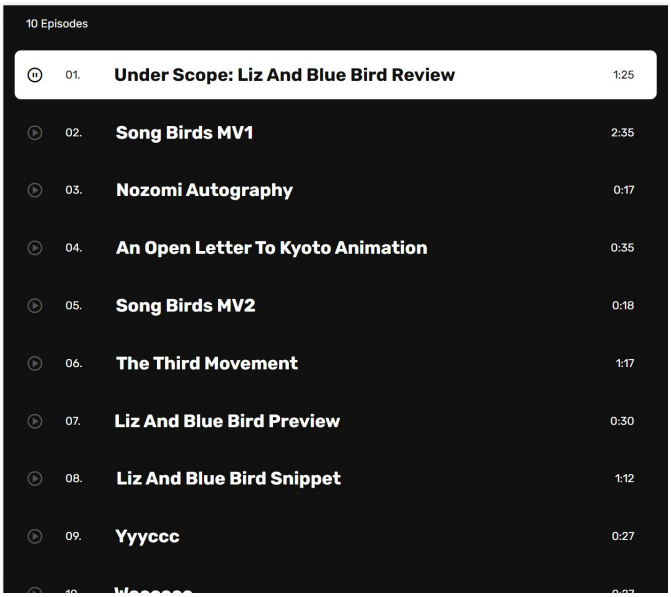
在上图中，上半部为原 JSON 文件，下半部为清理后的 JSON 文件，用 JS 中 parse 解析即可。

手动添加元素：

```
    var myList = document.querySelector(".videos");

    for (var i = 0; i < cv_list.length - 2; i++) {
      var newV = document.createElement("div");
      let e = i + 3;
      if (e < 10){
        e = '0' + e;
      }
      newV.innerHTML = '<i class="bx bx-play-circle"></i><i class="bx bx-pause-circle pause"></i><p class="numericalNumber">'+ e + '.</p><h3 class="title">title title title</h3><p class="time">3:46</p>'
      newV.classList.add("video");
      www = i + 2
      pt = "video-" + www;
      console.log(pt)
```

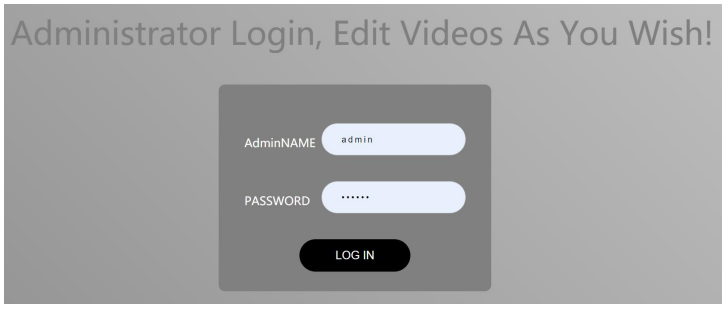
```
14.     newV.classList.add(pt);
15.
16.     myList.appendChild(newV);
17.   }
```

在主界面中视频文件列表正常显示，与后端数据库条目对应。



3.4.4 管理员登录

在管理员登录与正常游客登陆虽然设置了两个不同界面，但其登录的操作相同，仅为路由的跳转不同。



其后端对应表单为

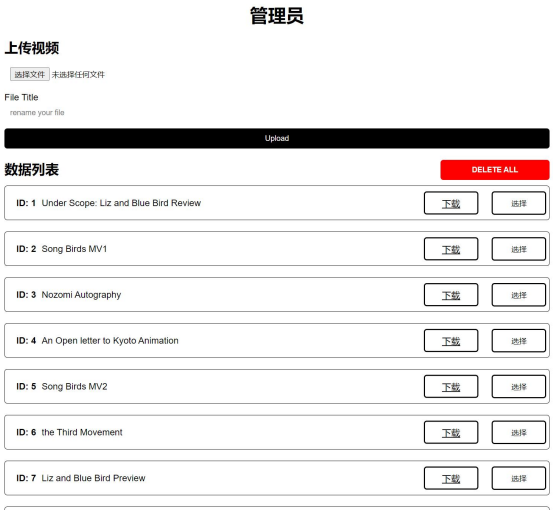
```
1.    class AdminTable(models.Model):
2.        admin_id = models.IntegerField(primary_key=True)
3.        admin_name = models.CharField(max_length=45, blank=True, null=Tru
    e)
```

```
        admin_psw = models.CharField(max_length=45, blank=True, null=True
    )

        class Meta:
            managed = False
            db_table = 'admin_table'
```

对应 MySQL 数据库表为

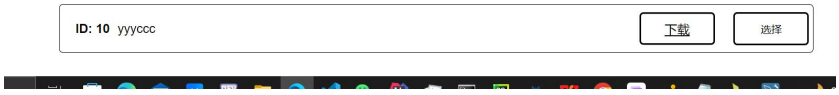
	admin_id	admin_name	admin_psw
▶	1	admin	114514
*	NULL	NULL	NULL

3.4.5 管理员删除视频



管理员主页面中可以同时选择多个项目进行删除，点击删除后页面跳转至 deleted.html 中，显示文件删除成功。

点击 return to manager page 返回管理员界面，此时界面已经成功删除条目



同时数据库中同样将对应条目删除

video_id	video_name
1	Under Scope: Liz and Blue Bird Review
2	Song Birds MV1
3	Nozomi Autography
4	An Open letter to Kyoto Animation
5	Song Birds MV2
6	the Third Movement
7	Liz and Blue Bird Preview
8	Liz and Blue Bird Snippet
10	yyyccc
NULL	NULL

1. JS 部分

通过为选中的条目添加类属性 selected，可以在点击删除时判断哪些元素被筛选到，并将其序列信息返回给后端。

```
    // 获取表单和列表元素
          const list = document.getElementById("list");

          // 添加数据的函数
          function addData(name, description) {
            // 删除按钮点击事件
            deleteBtn.addEventListener("click", () => {
              li.classList.add("selected");
            });
          }
```

点击 DELTE ALL 按钮，JS 检测哪些条目具有 selcted 属性，计算体哦啊木序列号后通过 AJAX 将请求发送给后端对应 URL

```
    deleteAll = document.querySelector(".all-delete");
          toBeDeleted = []
          deleteAll.addEventListener("click", () => {
              selectedItems = document.querySelectorAll(".list-item");
              for (let i = 0;i < selectedItems.length; i++) {
```

```
                if (selectedItems[i].classList.contains("selected")) {
                    tar = parseInt(selectedItems[i].querySelector("strong").innerHTML.slice(3));
                    toBeDeleted.push(tar);
                }
            }
            console.log(toBeDeleted)


            $.ajax({
              url: "../deleteItems/",
              type: "POST",
              data: { itemsToDelete : toBeDeleted },
              success: function (message) {
                  window.location.replace("../deleted/");
              },
              error: function () {

              },
            });
        })
```

获取对应序列号与获取该视频在当前列表中的位置不同，此处获取的应是后端数据库中对应的 id 号。

```
    deleteBtn.addEventListener("click", () => {
            if (li.classList.contains("selected")) {
                li.classList.remove("selected");
                deleteBtn.style.backgroundColor = '#fff';
                deleteBtn.style.color = '#000';
            } else {
              li.classList.add("selected");
              deleteBtn.style.backgroundColor = 'red';
              deleteBtn.style.color = 'white';
            }
          });
```

多选删除条目时进行样式变化

2. 后端处理

```
    def deleteItems(request):
        if request.method == 'POST':
            items_to_delete = request.POST.getlist('itemsToDelete[]')
            for item_id in items_to_delete:
                # 获取指定的记录并删除
                item = get_object_or_404(VideoName, video_id=item_id)
```


装 订 线

```
                item.delete()
            success = True
            data = {'message': "SUCCESS"}
        else:
            success = False
            data = {'message': "FAIL"}
        return JsonResponse(data, status=200 if success else 400)


    def deleted(request):
        return render(request, "deleted.html")
```

后端获取到 AJAX 请求后在数据库中删除对应条目。

装

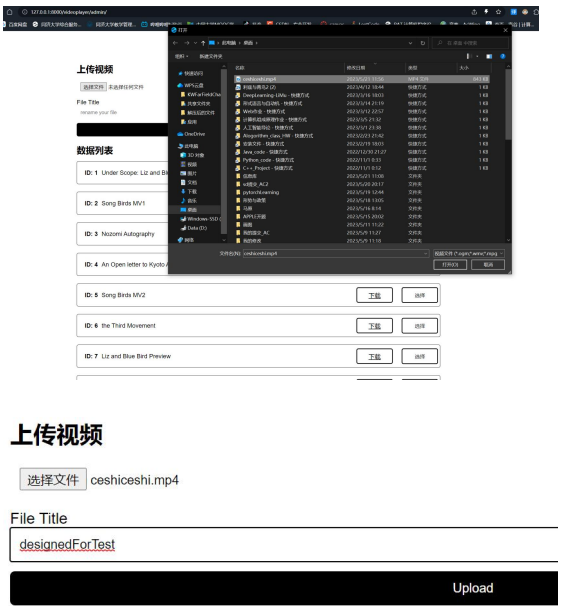
3.5 拓展功能实现

3.5.1 管理员在线上传视频

订

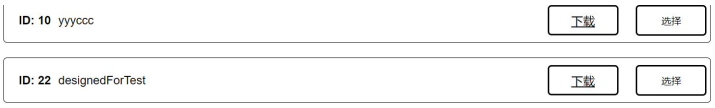
在管理员界面，点击选择文件，选择本地要上传的视频文件，指定上传的文件名称（显示在播放列表中），点击 Upload 上传。

线

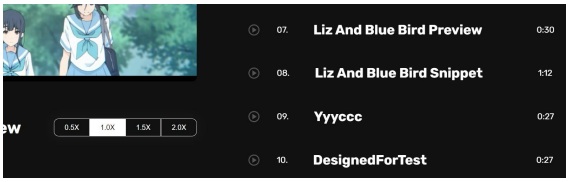


点击上传后如果上传成功则跳转到上传成功显示页面，点击返回按钮返回到管理员界面。

在管理员界面中已经出现上传文件



登录主界面，观看列表中也已经出现响应条目



1. HTML 部分

```
    <h2>上传视频</h2>
          <form action="/videoplayer/uploaded/" method="post" enctype="multipart/form-data">
              {% csrf_token %}
              <input type="file" name="video_file" accept="video/*">
              <label>File Title</label><input name="fileRename" placeholder="rename your file"/>
              <input type="submit" value="Upload">
          </form>
```

同样是表单形式提交，但在表单处需要设置 enctype 为 multipart/form-data，同时在进行 input 输入时需要限制输入文件为一个视频文件，在 input 的标签 type 设置为 file 后，需要在 accept 字段设置为 video/保证上传文件为视频文件。

2. JS 部分

```
    const li = document.createElement("li");
            li.classList.add("list-item")
            const deleteBtn = document.createElement("button");
            deleteBtn.innerText = "选择";
            deleteBtn.classList.add("delete-btn");

            const downBtn = document.createElement("a");
            downBtn.innerText = "下载";
            downBtn.style.display = "inline-box";
```

```
            downBtn.classList.add("down-btn");
            downBtn.setAttribute('download', '');

            // 添加数据到 li 元素中
            li.innerHTML = `<strong>ID: ${cv_list[i].video_id} </strong>
<span>${cv_list[i].video_name}</span>`;
            li.appendChild(downBtn);
            li.appendChild(deleteBtn);
            sq = cv_list[i].video_id;
            file_name = "video" + sq + ".mp4";
            downBtn.href = '../static/videos/'+file_name;

            // 将 li 元素添加到列表中
            list.appendChild(li);
```

加载数据库数据，动态添加视频列表内容，其中填充使用 js 模板串

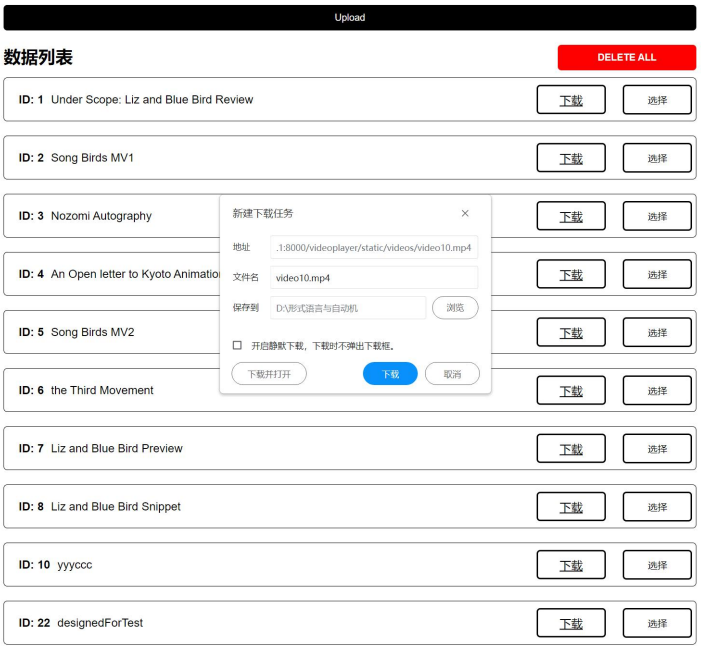
3. 后端处理

```
    def upload_video(request):
        if request.method == 'POST':

            video_file = request.FILES['video_file']
            video_data = video_file.read()

            rename = request.POST.get("fileRename", '')

            n = len(os.listdir(r"..\videoplayer\static\videos"))

            video_uploaded = VideoName(video_id=n + 1, video_name=rename)
            video_uploaded.save()

            new_filename = "video" + str(n + 1) + ".mp4"

            dir_path = r'..\videoplayer\static\videoss'

            with open(os.path.join(dir_path, new_filename), 'wb') as f:
                f.write(video_data)
            return render(request, "uploaded.html")
        else:
            return HttpResponse("Failed")
```

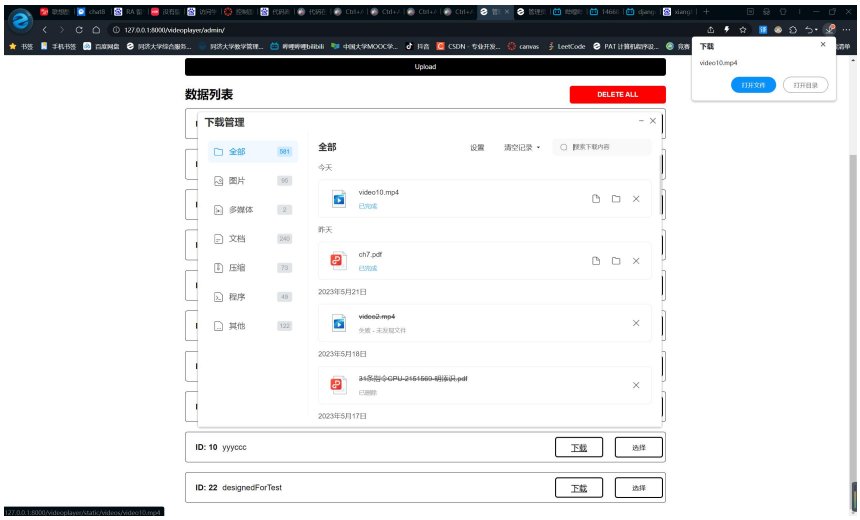
后端获取到文件名称之后由字符串拼接获取文件名称和路径，在对应位置写入视频文件。

3.5.2 管理员在线下载视频

进入管理员界面，点击对应视频的下载按钮



点击第十个视频文件进行下载，弹出下载框指定路径进行下载



开始下载到本地，本地查看

点击播放，可以正常播放。

装

1. html 部分

使用 a 超链接标签，浏览器可以进行识别对应路径资源并调用下载功能。，由于时动态添加条目，多以实现部分在 JS 中。

订

2. JS 部分

线

```
const downBtn = document.createElement("a");
downBtn.innerText = "下载";
downBtn.style.display = "inline-box";
downBtn.classList.add("down-btn");
downBtn.setAttribute('download', '');
downBtn.addEventListener("click", () => {
            $.ajax({
            url: "../downItems/",
            type: "POST",
            data: { itemsToDownload : mess },
            success: function (message) {

            },
            error: function () {

            },
          });
        });
```

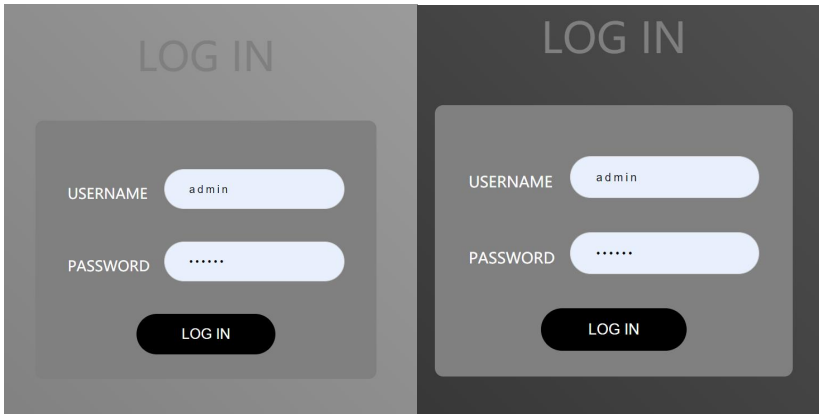
a 标签添加 download 属性后浏览器可以识别并进行下载功能调用。如果视频实体存储在数据库，使用 AJAX 的方式可以传递给后端进行处理。

3. 后端处理下载请求

```
    def downItems(request):
        if request.method == 'POST':
            items_to_download = request.POST.getlist('itemsToDownload[]')
            # print(int(items_to_download[0]))
            sq = items_to_download[0]
            file_name = r"video" + sq + ".mp4"
            # file_path = r"../static/videos/" + file_name
            file_path = os.path.join(settings.BASE_DIR, 'videoplayer\\static\\videos\\'+file_name)
            # D:\Web 作业\实验4-前后端\videoplayer\videoplayer\static\videos
            video_file = open(file_path, 'rb')
            response = FileResponse(video_file)
            response['Content-Disposition'] = 'attachment;filename="%s"' % file_name
        else:
            response = {"message": "OK"}
        print(response)
        return response
```

3.5 界面优化

登录界面使用动态渐变效果

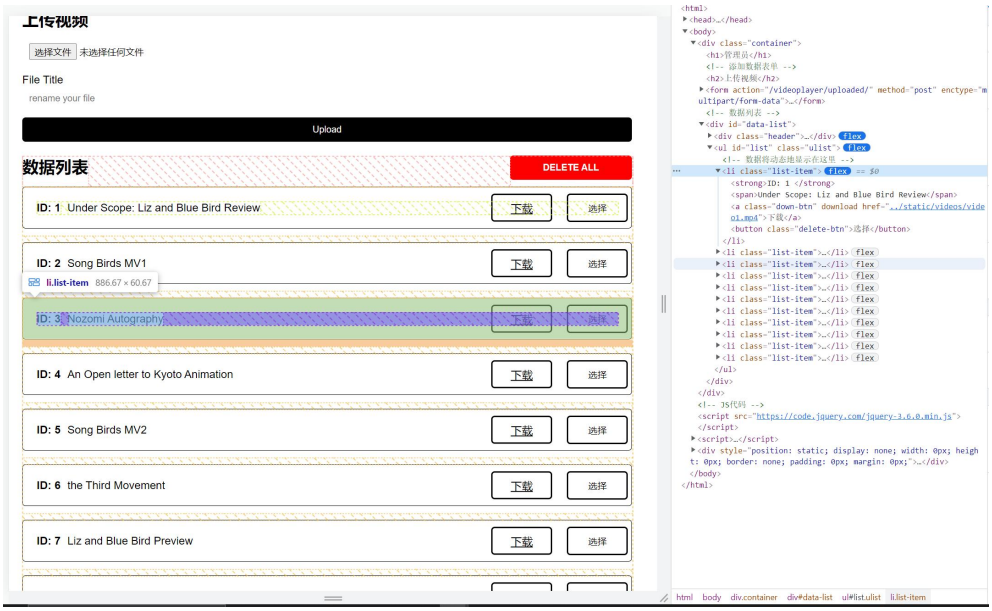


使用 CSS 动画效果实现背景动态变化

```
    html {
            background: linear-gradient(45deg, black, white);
            background-size: 400% 400%;
            animation: gradient 20s ease infinite;
         }
```

```
6.
7.          @keyframes gradient {
8.            0% {
9.              background-position: 0% 50%;
10.           }
11.           50% {
12.             background-position: 100% 50%;
13.           }
14.           100% {
15.             background-position: 0% 50%;
16.           }
17.         }
```

管理员界面 flexbox 布局



4. 实验总结

通过服务器端编程，我熟悉了后端服务的大致框架，了解了后端配置路由、安全、业务处理的方式，了解了如何进行前后端衔接、后端与数据库进行连接交互，完成了页面构建、中台连接、后端处理、数据库交互的流程。同时我熟悉了掐断 JS 代码的编写以及快速构建 HTML 页面过程。了解了 Django 框架的特性、功能、语法等，对 Web 应用开发形成了大体认知。