

浙江大学



课程名称: B/S 体系软件设计

项目名称: 图像标注网站

指导老师: 胡晓军

学 院: 计算机科学与技术

个人信息: 郑无恙 3180101821 计科 1803

一、项目简介

1.1 运行环境

本项目采用了 Flask+Restful+SQLAlchemy+jQuery 进行综合开发，实现了一个能够进行图像标注并发布相应任务的网站。

本人的运行环境为：

- Windows 11 21H2 (Build 22000.376)
- Python 3.10.0

1.2 如何安装

在项目目录上，执行 `python -m venv venv` 命令，创建当前目录上的一个虚拟环境。接下来，

如果使用的是 Windows 系统，执行 `venv\Scripts\activate` 命令，启用虚拟环境。

如果使用的是 Linux 系统，执行 `. venv/bin/activate` 命令，启用虚拟环境。

然后执行 `pip install -r requirements.txt` 安装项目的所有第三方包依赖。

1.3 如何使用

启用虚拟环境后，执行 `python run.py` 启动 flask 服务器。

二、总体架构及功能设计

2.1 总体架构

本项目采用了 REST (Representational State Transfer) 架构风格进行实现。Restful 是一种网络应用程序的设计风格 and 开发方式，基于 HTTP，可以使用 XML 格式定义或 JSON 格式定义。RESTFUL 适用于移动互联网厂商作为业务接口的场景，实现第三方 OTT 调用移动网络资源的功能，动作类型为新增、变更、删除所调用资源。

基于 Restful 的设计原则，本项目的所有关键函数都存放在 `/api/resource/` 目录内。为了便于视图函数的书写，把整个项目的入口变得更

加简洁，并把函数代码归类到一起，项目使用了蓝图注册的方式将关键函数全部封装在/api/__init__.py 文件内。

本项目主要分为以下 6 个模块：

- 1. **图片/视频上传模块：**实现批量图片/视频的上传，将这些上传的图片保存到当前用户创建的最近数据集目录内；
- 2. **页面获取模块：**当用户发送 URL 请求到服务器后，渲染相对应的页面并返回给用户；
- 3. **静态文件获取模块：**当用户请求页面中的图片资源（个人头像、数据集图片、数据集封面等）时，返回服务器中对应的静态资源给用户；
- 4. **JavaScript 操作模块：**当用户执行页面的 JavaScript 后，会发送带有数据的 POST 至服务器，服务器需对这些 POST 做出相应并修改对应的静态资源数据；
- 5. **用户管理模块：**负责管理用户的注册、登录及登出等功能；
- 6. **图像标注任务模块：**渲染已发布数据集以及图像标注页面。

2.2 主要功能设计

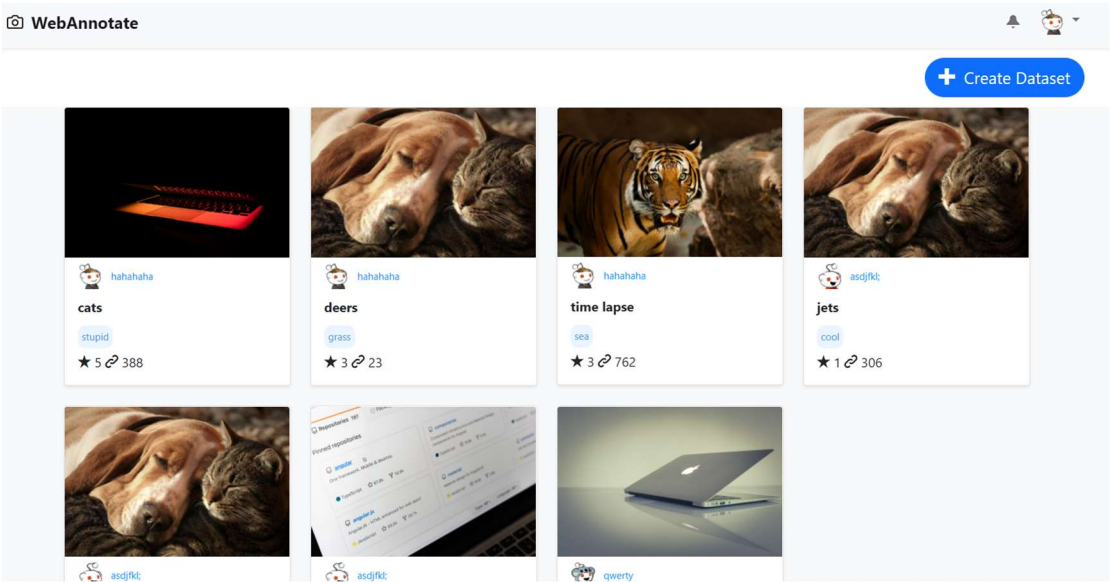
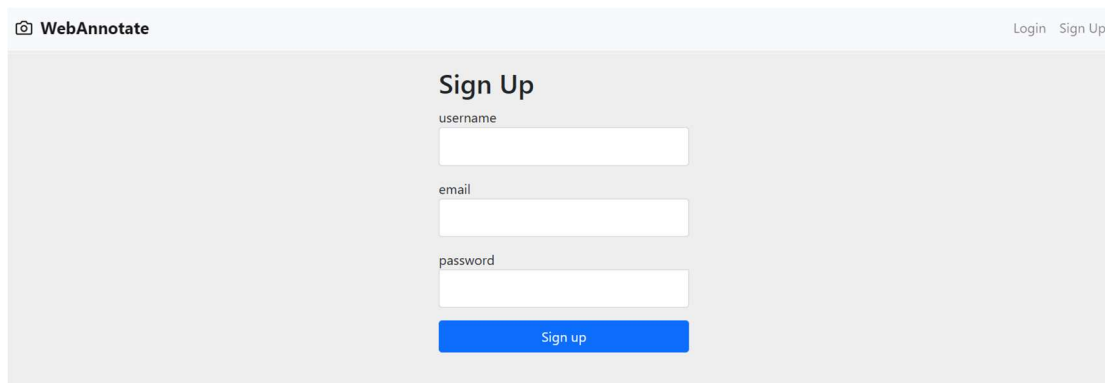


图 1：首页页面设计

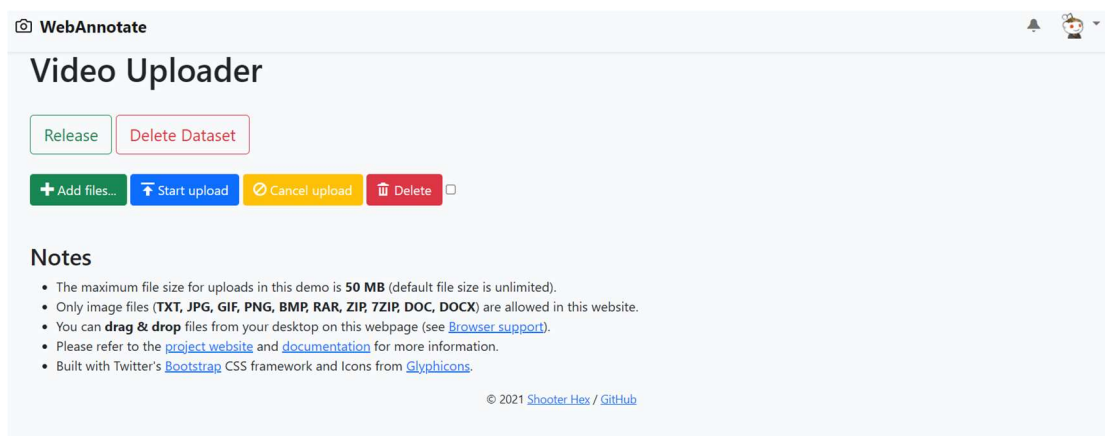
图 1 为本项目的首页，可以看到首页上显示了一些其他用户已经发布的数据集，以及它们各自的标题、标签和收藏及被引数量。首页右上方显示当前为用户处于登录状态。



The image shows the 'Sign Up' page of the WebAnnotate application. At the top left is the 'WebAnnotate' logo, and at the top right are links for 'Login' and 'Sign Up'. The main heading is 'Sign Up'. Below it are three input fields labeled 'username', 'email', and 'password'. At the bottom is a blue button labeled 'Sign up'.

图 2：注册页面设计

图 2 显示了用户未登录时上方栏的显示状态，同时也展示了用户在注册帐号时的表格输入栏位。



The image shows the 'Video Uploader' page of the WebAnnotate application. At the top left is the 'WebAnnotate' logo, and at the top right are a bell icon, a user profile icon, and a dropdown arrow. The main heading is 'Video Uploader'. Below it are two buttons: 'Release' (green) and 'Delete Dataset' (red). Below these are four buttons: '+ Add files...' (green), 'Start upload' (blue), 'Cancel upload' (yellow), and 'Delete' (red) with a trash icon. Below the buttons is a section titled 'Notes' with a bulleted list of information. At the bottom right is the copyright notice '© 2021 Shooter Hex / GitHub'.

图 3：视频上传页面设计

图 3 为用户上传视频的页面，一共有添加视频文件、开始上传、取消上传、删除、全选图片、发布数据集及删除数据集七个按钮。实现了提取视频关键帧及图片管理功能。

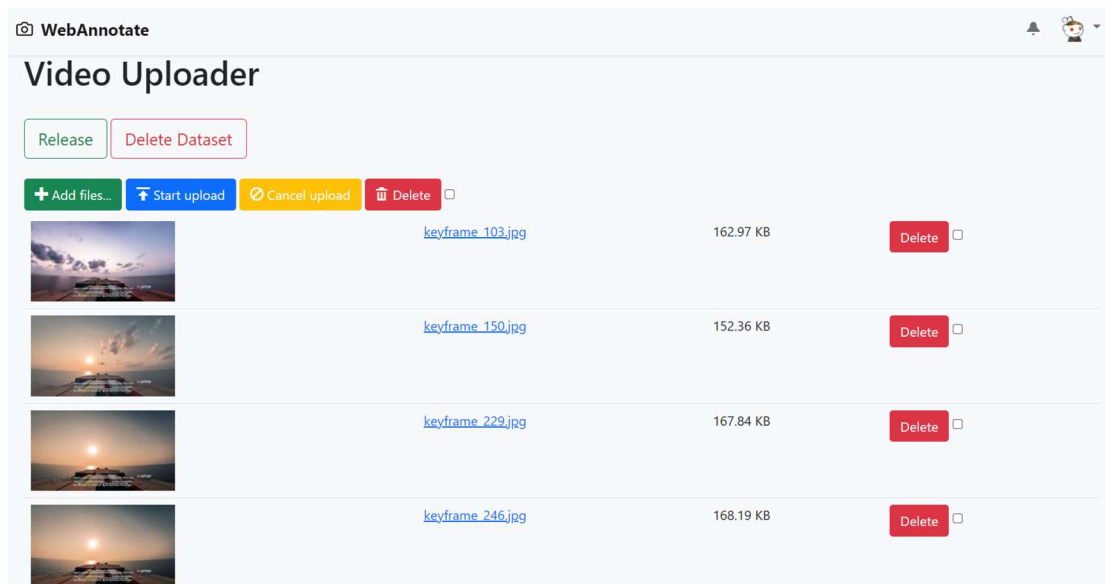


图 4：提取关键帧后的页面

用户在完成视频的上传后，刷新页面，即可看到提取关键帧后的图片。

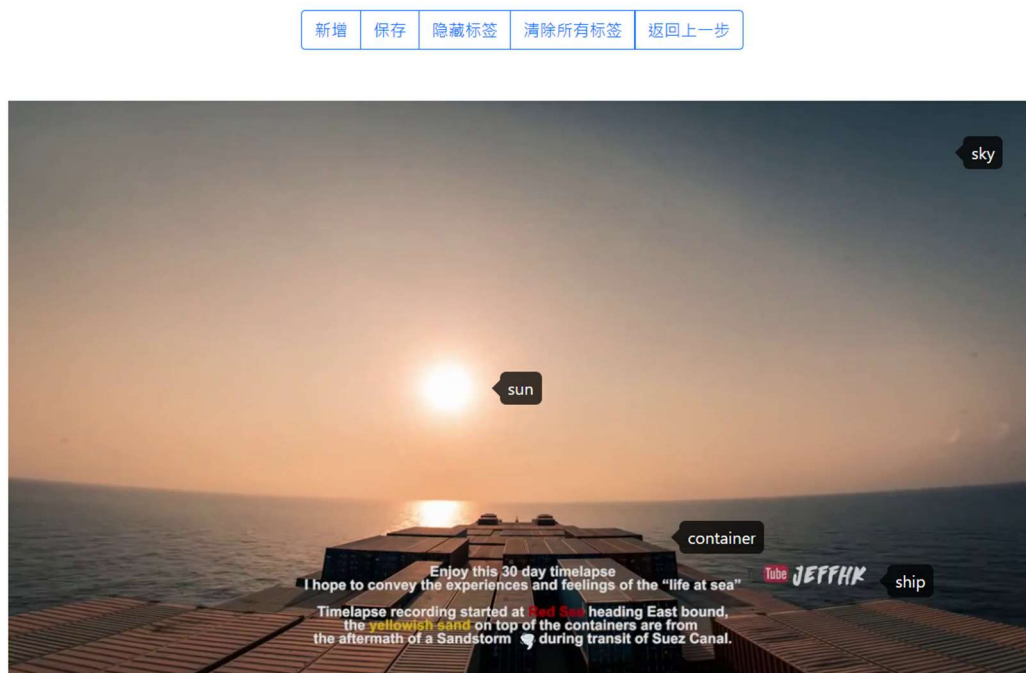


图 5：图像标注页面

图 5 显示了本项目的标注界面，能在图片上新增、保存、隐藏或清除标签。

三、关键数据结构/算法

3.1 用户注册及登录功能

```
class LoginForm(FlaskForm):
    username = StringField('username', validators=[InputRequired(),
                                                    Length(min=6, max=15)])
    password = PasswordField('password', validators=[InputRequired(),
                                                    Length(min=6, max=80)])
    remember = BooleanField('remember me')

class RegisterForm(FlaskForm):
    email = StringField('email', validators=[InputRequired(),
                                              Email(message='Invalid email'),
                                              Length(max=50)])
    username = StringField('username', validators=[InputRequired(),
                                                    Length(min=6, max=15)])
    password = PasswordField('password', validators=[InputRequired(),
                                                    Length(min=6, max=80)])
```

图 6：用户注册及登录类

图 6 显示了用户注册及登录的表格项，这里使用了 flask_login 模块对这些项做初始化。当用户提交表格时，flask 会利用这些初始化好的 validators 对判断用户提交的表单数据是否合法。如果不合法，则在页面上显示相关栏位输入不合法的原因。

```
class Login(Resource):
    def get(self):
        form = LoginForm()

        if form.validate_on_submit():
            user = User.query.filter_by(username=form.username.data).first()
            if user:
                if check_password_hash(user.password, form.password.data):
                    login_user(user, remember=form.remember.data)
                    return redirect("/")

            return make_response(render_template_string('<h1>Invalid username or password</h1>'))

        return make_response(render_template('login.html', form=form))

    def post(self):
        return self.get()
```

图 7：用户登录模块

图 7 为项目的登录模块。当用户提交表单后，后台会在服务器端数据库查询当前是否存在该用户。若该用户存在，则将用户提交的密码进行散列运算并判断散列值是否等于数据库中保存的数值。如果相等，则使当前用户登录，并重定向回首页。如果不相等，则返回 “Invalid username or password” 的页面。

```
class Signup(Resource):
    def get(self):
        form = RegisterForm()
        status_code = 0

        if form.validate_on_submit():
            # check database if same user name/email exists
            user = User.query.filter_by(username=form.username.data).first()
            email = User.query.filter_by(email=form.email.data).first()
            if user:
                status_code = 1
            elif email:
                status_code = 2
            else:
                hashed_password = generate_password_hash(form.password.data,
                                                            method='sha256')
                new_user = User(username=form.username.data,
                                email=form.email.data,
                                password=hashed_password)
                db.session.add(new_user)
                db.session.commit()
```

图 8：用户注册模块（一）

当用户进行注册的时候，后台的操作和登录模块基本类似。首先也是判断后台数据库是否已经存在用户提交表单中的用户名或邮箱地址。如果都不存在，则计算用户提交表单中密码的散列值，并将用户名、邮箱地址以及密码的散列值保存到后台数据库上。

```

avatar_index = random.randrange(10) + 1
if os.path.exists('./static/json/user_data.json'):
    with open("./static/json/user_data.json",
              'r+') as json_fp:
        data = json.load(json_fp)
        data['user'].append({
            'id': new_user.id,
            'name': new_user.username,
            'avatar': avatar_index
        })
        json_fp.seek(0, 0)
        json.dump(data, json_fp)
else:
    data = {'user': []}
    data['user'].append({
        'id': new_user.id,
        'name': new_user.username,
        'avatar': avatar_index
    })
    with open('./static/json/user_data.json',
              'w') as json_fp:
        json.dump(data, json_fp)

return make_response(render_template_string('<h1>New user has been created!</h1>'))

```

图 9：用户注册模块（二）

接下来，使用随机数生成一个 1 到 10 之间的数值，作为系统分配给用户头像的索引值。接着判断后台是否已经保存了用户的 json 文件。假如这个 json 文件已经创建，则仅需在这个 json 文件上附加新的用户数据。完成附加操作后，将修改好的 json 数据重新保存到文件中。如果 json 仍未创建，则初始化当前用户的 json 数据并保存到新创建的这个 json 文件里面。完成上述操作后，返回一个 “New user has been created!” 的页面。

3.2 视频关键帧提取功能

```
# save file to disk
uploaded_file_path = os.path.join(status.working_path,
                                   filename)
files.save(uploaded_file_path)

# get file size after saving
size = os.path.getsize(uploaded_file_path)

# return json for js call back
frame_extract(uploaded_file_path, status.working_path)

os.remove(uploaded_file_path)
result = uploadfile(name=filename, type=mime_type, size=size,
                    not_allowed_msg='视频关键帧已提取, 请重新刷新页面!',
                    is_refresh=True)

return {"files": [result.get_file()]}
```

图 10: 视频上传请求响应模块

在介绍视频关键帧提取算法前, 先对执行该算法前的操作进行介绍。当用户完成视频的上传后, 首先将这个视频暂存到一个文件中, 然后再调用关键帧提取算法。完成关键帧的提取后, 就把暂存的视频文件删除。最后再构造一个带有“视频关键帧已提取, 请重新刷新页面!”信息的 json 数据, 返回给上一级的请求者。这样就实现了完成视频上传并提取关键帧后, 页面更新一个关键帧提取完成的信息。

```

def frame_extract(videopath, dir):
    # smoothing window size
    len_window = 50

    # load video and compute diff between frames
    cap = cv2.VideoCapture(str(videopath))
    prev_frame = None
    frame_diffs = []
    frames = []
    success, frame = cap.read()
    i = 0
    while success:
        luv = cv2.cvtColor(frame, cv2.COLOR_BGR2LUV)
        curr_frame = luv
        if curr_frame is not None and prev_frame is not None:
            # logic here
            diff = cv2.absdiff(curr_frame, prev_frame)
            diff_sum = np.sum(diff)
            diff_sum_mean = diff_sum / (diff.shape[0] * diff.shape[1])
            frame_diffs.append(diff_sum_mean)
            frame = Frame(i, diff_sum_mean)
            frames.append(frame)
        prev_frame = curr_frame
        i = i + 1
        success, frame = cap.read()
    cap.release()

```

图 11: 计算帧间差分

现在正式介绍关键帧提取算法。首先使用 opencv 的 VideoCapture 函数读取视频的每一帧，并将色彩空间从 RGB 转换 LUV。接着计算当前帧与上一帧的差分，并计算整个差分的平均值，并将每一个帧的索引值保存到 frames 变量中。

```

# compute keyframe
keyframe_id_set = set()
diff_array = np.array(frame_diffs)
sm_diff_array = smooth(diff_array, len_window)
frame_indexes = np.asarray(argrelextrema(sm_diff_array, np.greater))[0]
for i in frame_indexes:
    keyframe_id_set.add(frames[i - 1].id)

```

图 12: 使用平均帧间差分强度局部最大值提取关键帧

接下来初始化一个 `keyframe_id_set` 的集合，并将上一步计算出来的差分值转换为 `numpy` 的数组。然后使用卷积核对这些差分值做平滑处理，达到一定的去噪效果，再计算平均帧间差分强度局部最大值的帧作为视频的关键帧。值得一提的是，这种方法的提取结果在丰富度上的表现更好一些，能均匀地提取视频中的关键帧。

```
# save all keyframes as image
cap = cv2.VideoCapture(str(videopath))
success, frame = cap.read()
idx = 0
while success:
    if idx in keyframe_id_set:
        name = "keyframe_" + str(idx) + ".jpg"
        img_dir = dir + '/images/' + name
        cv2.imwrite(img_dir, frame)
        UploadPicture.save_thumb(img_dir, name)
        keyframe_id_set.remove(idx)
    idx = idx + 1
    success, frame = cap.read()
cap.release()
```

图 13：保存关键帧数据

最后再次读入视频的每一帧，判断当前帧是否处于关键帧索引数组内。若是，则在用户当前创建的数据集中保存当前帧及其缩略图。完成对视频每一帧的遍历后，便完成了视频关键帧的提取。

3.3 数据集发布功能

```
function name_dataset_js() {  
    var name = prompt( message: "请输入您的数据集名称", _default: "");  
    var tag = prompt( message: "请输入一个数据集标签（如果无，直接点击OK）", _default: "")  
    if (name!=" " && name != null) {  
        $.ajax( url: {  
            url : "/name_dataset",  
            data :{'name' : name,  
                'tag': tag},  
            type : "POST",  
            success : function(data){  
                if (data == "200"){  
                    alert("命名成功");  
                    window.location.replace('/');  
                }  
                else{  
                    alert("命名失败");  
                }  
            }  
        })  
    }  
}
```

图 14: 命名数据集的 js 函数

图 14 为当前用户完成数据集的创建后，为数据集进行命名的 js 函数。首先浏览器会弹出两个窗口，分别要求用户提供数据集的名称及标签。然后调用 ajax 命令向后台的/name_dataset 这个 URL 发送数据中带有数据集的名称及标签的 POST 命令，如果后台返回了“200”，说明已经完成了数据集的命名。这个时候弹出一个“命名成功”的窗口，并重定向回首页。否则就弹出“命名失败”的窗口。

```

function delete_dataset_js() {
    if (confirm('您是否要删除当前仍未发布的数据集?')) {
        $.ajax( url: {
            url : "/delete_dataset",
            type : "POST",
            success : function(data){
                if (data == "200"){
                    alert('删除成功, 返回首页。');
                    window.location.replace('/');
                }
                else{
                    alert("删除失败! ");
                }
            }
        })
    }
}

```

图 15: 删除数据集的 js 函数

删除数据集的 js 操作与命名数据集基本一致，这个部分等到介绍后台部分代码时候再做详细说明。

```

class NameDataset(Resource):
    def get(self):
        name = request.form.get('name')
        tag = request.form.get('tag')
        uid = int(current_user.get_id()) - 1
        favorite_num = random.randrange(6)
        reference_num = random.randrange(2000) + 1
        cover_img = random.randrange(12) + 1

```

图 16: 命名数据集请求相应模块（一）

当用户发来命名数据集的 POST 请求后，后台提取出请求中的数据集名称及标签，并对数据集的收藏数、引用书以及使用封面的索引用随机数做初始化。

```

with open("./static/json/user_data.json", 'r+') as json_fp:
    data = json.load(json_fp)
    if 'datasets' in data['user'][uid].keys():
        if tag != '':
            dataset = {'dataset_id': len(data['user'][uid]['datasets']) + 1,
                        'dataset_name': name,
                        'dataset_cover': cover_img,
                        'dataset_tag': tag,
                        'dataset_fav': favorite_num,
                        'dataset_ref': reference_num}
        else:
            dataset = {'dataset_id': len(data['user'][uid]['datasets']) + 1,
                        'dataset_name': name,
                        'dataset_cover': cover_img,
                        'dataset_fav': favorite_num,
                        'dataset_ref': reference_num}

```

图 17: 命名数据集请求相应模块 (二)

接着打开后台的 json 文件，判断用户是否曾经创建过数据集。如果曾创建过，则判断用户是否为数据集提供了标签，并做相对应的 json 数据初始化。

```

else:
    data['user'][uid]['datasets'] = []
    if tag != '':
        dataset = {'dataset_id': 1,
                    'dataset_name': name,
                    'dataset_cover': cover_img,
                    'dataset_tag': tag,
                    'dataset_fav': favorite_num,
                    'dataset_ref': reference_num}
    else:
        dataset = {'dataset_id': 1,
                    'dataset_name': name,
                    'dataset_cover': cover_img,
                    'dataset_fav': favorite_num,
                    'dataset_ref': reference_num}

```

图 18: 命名数据集请求相应模块 (三)

如果这次是用户第一次创建他/她自己的数据集，则为当前所创建的数据集 id 分配为 1，表明这是用户创建的第一个数据集。

3.4 图片标注功能

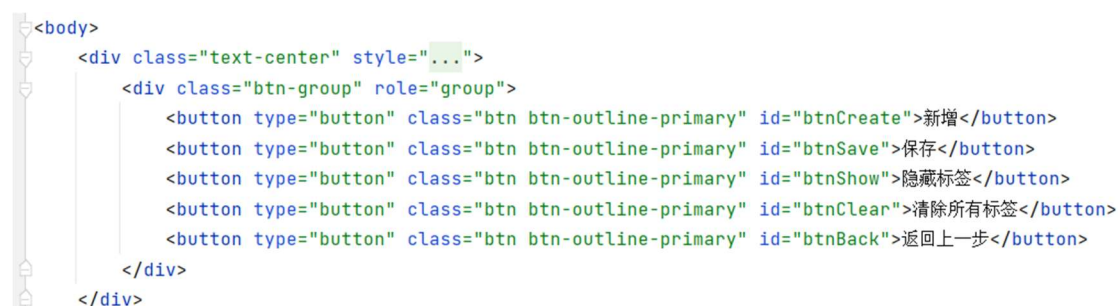


图 19：图像标注页面按钮

图19为图片标注页面顶部展示的六个按钮，它们会各自触发相应的js操作。

```


<script src="../../static/js/jquery.min.js"></script>
<script src="../../static/js/json2.min.js"></script>
<script src="../../static/js/jquery.contextMenu.min.js"></script>
<script src="../../static/js/jquery.image-label.js"></script>
```

图 20：显示待标注图片

接着利用 jinja2 语法调用后台提供的图片 URL 地址，显示当前待标注图片。最后再引入 jQuery、json2 等 js 库。

```
//初始化
$('img').imageLabel();
//加载数据
var _arr = JSON.parse(localStorage.getItem('labelArr'));
$('img').imageLabel('loadData', {
  data: _arr
});
//create
$('#btnCreate').click(function(){
  $('img').imageLabel('create');
});
//save
$('#btnSave').click(function(){
  var data = $('img').imageLabel('getData');
  localStorage.setItem('labelArr', JSON.stringify(data));
});
```

图 21：按钮点击响应处理（一）

从图 21 可以看到，网页 js 在初始化的时候会调用 json 解析 localStorage 中的“labelArr”项，并保存到_arr 变量中。当用户点击创建标签按钮后，调用后台 js 库的“create”方法；当用户点击保存标签按钮后，调用后台 js 库的“getData”方法，并将返回的数据保存到 data 变量中，再调用 localStorage 将 json 化的 data 变量保存到本地。

```
$('#btnClear').click(function(){
    localStorage.clear();
    location.reload();
});
$('#btnBack').click(function(){
    history.back()
})
//toggle
$('#btnShow').click(function(){
    var isShow = $(this).attr('_flag');
    isShow = typeof(isShow)=='undefined'?1:isShow;
    if (isShow && isShow==1){
        $(this).text('展示标签');
        $('img').imageLabel('hide');
        $(this).attr('_flag', 0);
    }else{
        $(this).text('隐藏标签');
        $('img').imageLabel('show');
        $(this).attr('_flag', 1);
    }
});
```

图 22：按钮点击响应处理（二）

当用户点击清除标签按钮后，清除 localStorage 中的所有数据，并刷新页面；当用户点击返回上一步按钮后，调用 history 中的 back 方法返回上一步；当用户点击隐藏或显示标签按钮后，利用 isShow 变量控制 imageLabel 中“hide”和“show”方法的调用。


```

$.contextMenu({
  selector: '.kbs-label-area',
  callback: function(key, options) {
    var e = window.event || arguments[0];
    $('img').imageLabel('create', {
      top: $('.context-menu-list')[0].offsetTop || e.clientY,
      left: $('.context-menu-list')[0].offsetLeft || e.clientX
    });
  },
  items: {
    "create": {name: "新增标签"}
  }
});

```

图 23：按钮点击响应处理（三）

最后介绍新增标签的 js 操作，首先将 DOM 的 selector 选取到 kbs-label-area 类上，并初始化一个回调函数以及一个项目，这样就实现了新增标签的功能。

四、项目总结

在本次的项目实战中，我积累了一些全栈开发上的经验。尽管自己仅仅达到入门水平，但我还是学会了很多前后端相关的知识。我在这次大作业使用 flask 的原因是我自己对 Python 比较熟悉，觉得使用基于 Python 的框架对我来说上手可能会更快些，在一次次的摸索中实现了一个基于 Python 搭建的网站。虽然自己未来并没有从事前端行业的打算，但我相信自己积累过的前端经验能在以后某些项目的实现中为我提供一些帮助，毕竟这也是许多程序员都掌握的一项基本技能。