Lab2: Image Retrieval

> @author 2154298 Ying Wang

Installation

- To run the code, you should also download and unzip database.zip in `server`.
    1. You can easily download the dataset from the provided link and extract it. Then, place the extracted folder in the "server/database" directory.
    2. Alternatively, you can use the command line to download and install it.
    - navigate to the directory where you want to download and extract the file. Then, enter the following command:

      ```
      wget https://anjt.oss-cn-shanghai.aliyuncs.com/database.zip
      ```

    - use the following command to extract the file:

      ```
      unzip database.zip
      ```

    - Note that before running these commands, make sure you have installed the `wget` and `unzip` tools.
- After installing the essential package, you can run the code as follows:

  ```
  cd server
  python image_vectorizer.py
  ```

- Start the server as follows:

  ```
  python rest-server.py
  ```

  Then you can visit the website: http://127.0.0.1:5000/

Project Structure

```
root folder
|  neighbor_list_recom.pickle
|  requirements.txt
|  __init__.py
|
└─server
    |  image_vectorizer.py
    |  neighbor_list_recom.pickle
    |  rest-server.py
    |  saved_features_recom.txt
    |  search.py
    |
    ├─database
    |  |
    |  ├─dataset
```

```
|   |
|   └─tags
|
|
├─imagenet
|       classify_image_graph_def.pb
|
├─static
|   |   main.js
|   |   style.css
|   |
|   ├─collection
|   |
|   ├─images
|   |
|   └─result
|
├─templates
|        collection.html
|        main.html
|
└─uploads
```

Task Requirement

The requirements of an image search task is as follows:

- **Relevant Image Collection**

  A relevant and well-curated collection of images is essential for an effective image search task. The images should be labeled with descriptive metadata, such as keywords or tags, to enable efficient searching.

- **Efficient Search Algorithm**

  An efficient search algorithm is required to quickly and accurately retrieve the relevant images from the collection. The search algorithm may be based on various techniques such as visual feature extraction, deep learning, or machine learning.

- **User Interface**

  A user-friendly interface is essential for an image search task. The interface should allow users to easily input their search queries, browse and view the search results, and provide relevant feedback on the search results.

- **High Quality Image**

  The quality of the images in the collection should be high to enable accurate and detailed visual analysis.

- **Scalability**

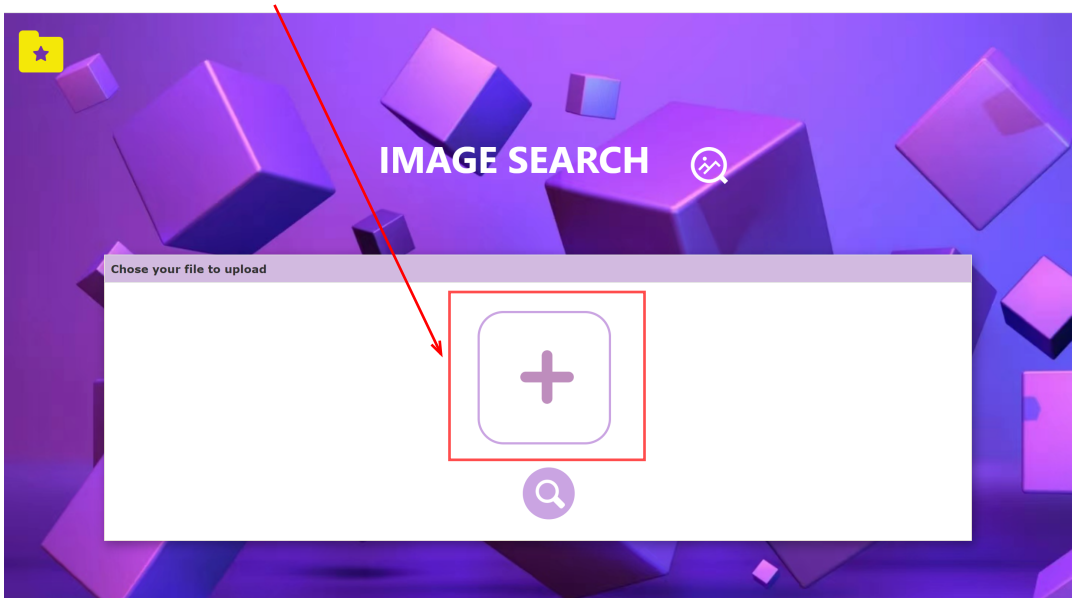
  The image search system should be scalable to handle large volumes of images and queries without compromising performance or accuracy.

- **Security**

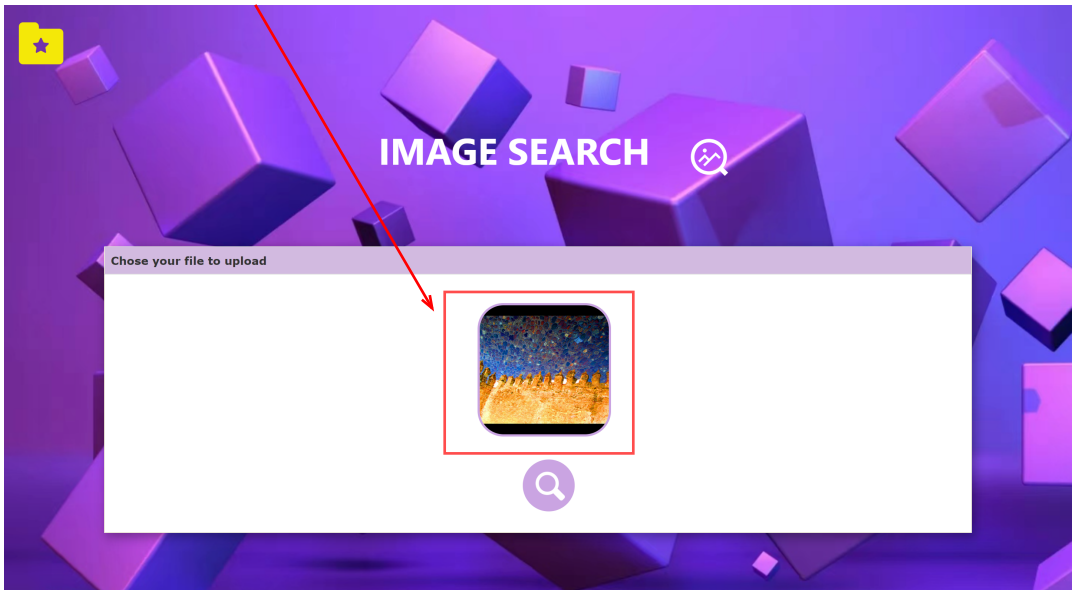
Security is also an important consideration for an image search task. The system should be designed to protect the privacy and security of the images and user data.

Functionality

- ## Formulation
  - It contains an input box to upload an image:

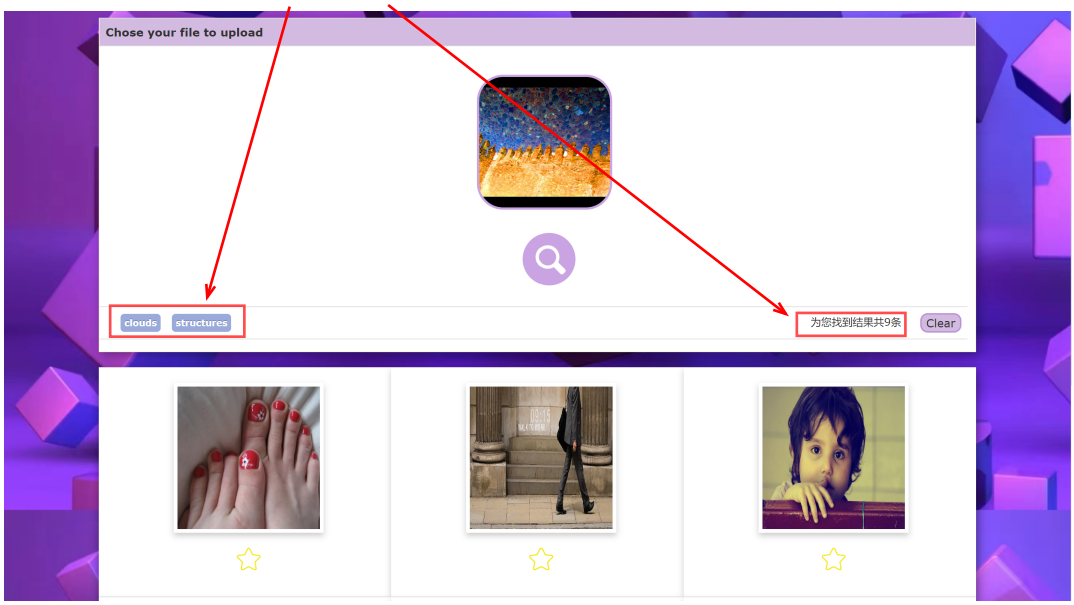


  - Users can preview the query image in the searching window:



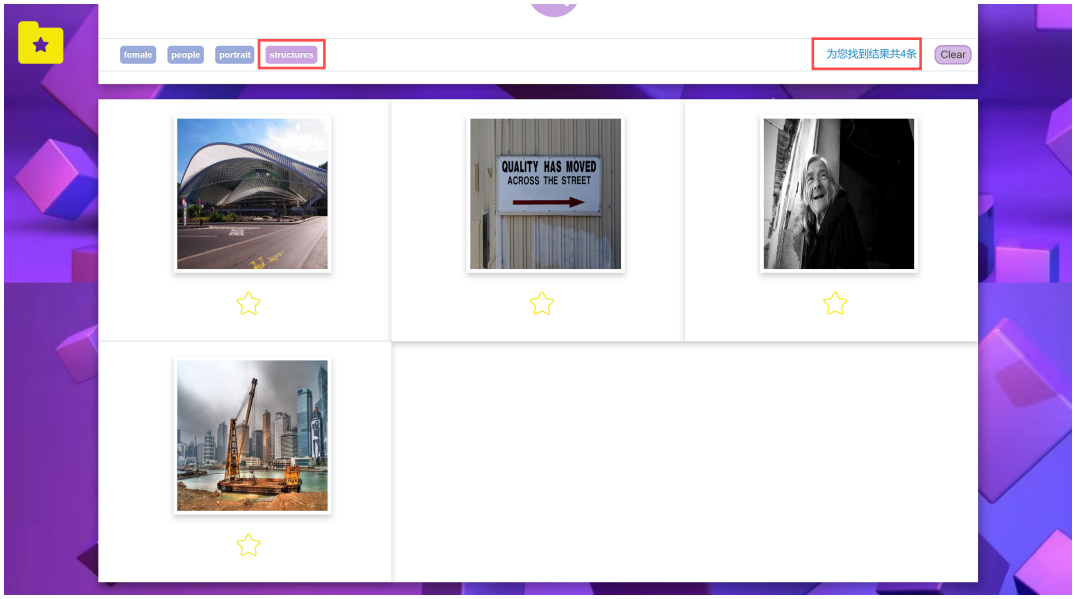
- ## Initiation of action
  - It has a search button, click the button for results:

- **Review of results**
  - Provide an overview of the results, the overview displays the number of search results

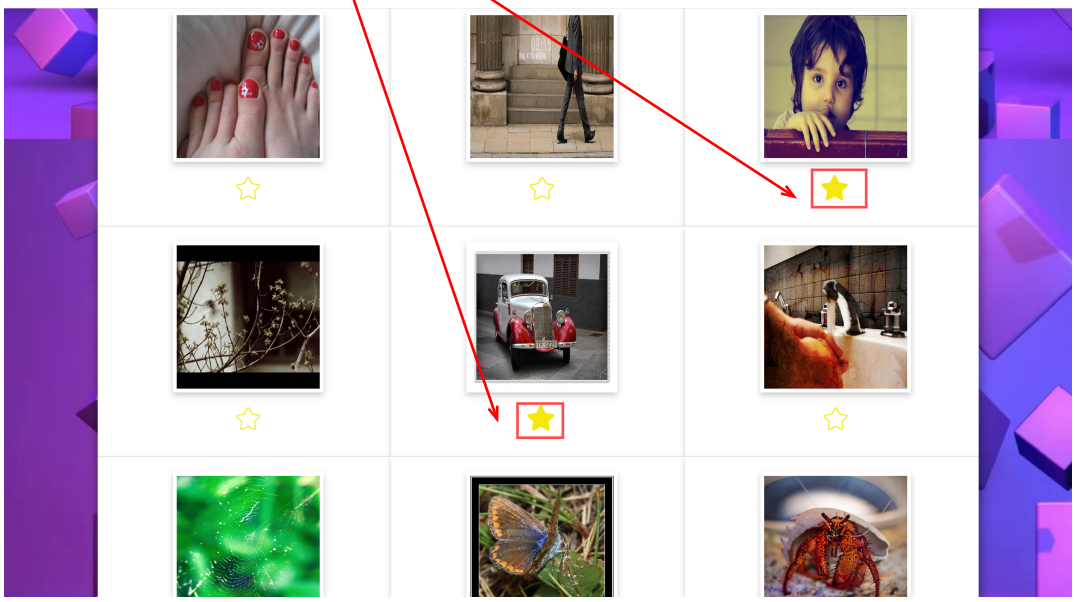


- **Refinement**
  - Allow changing search parameters （select certain tag when reviewing results）

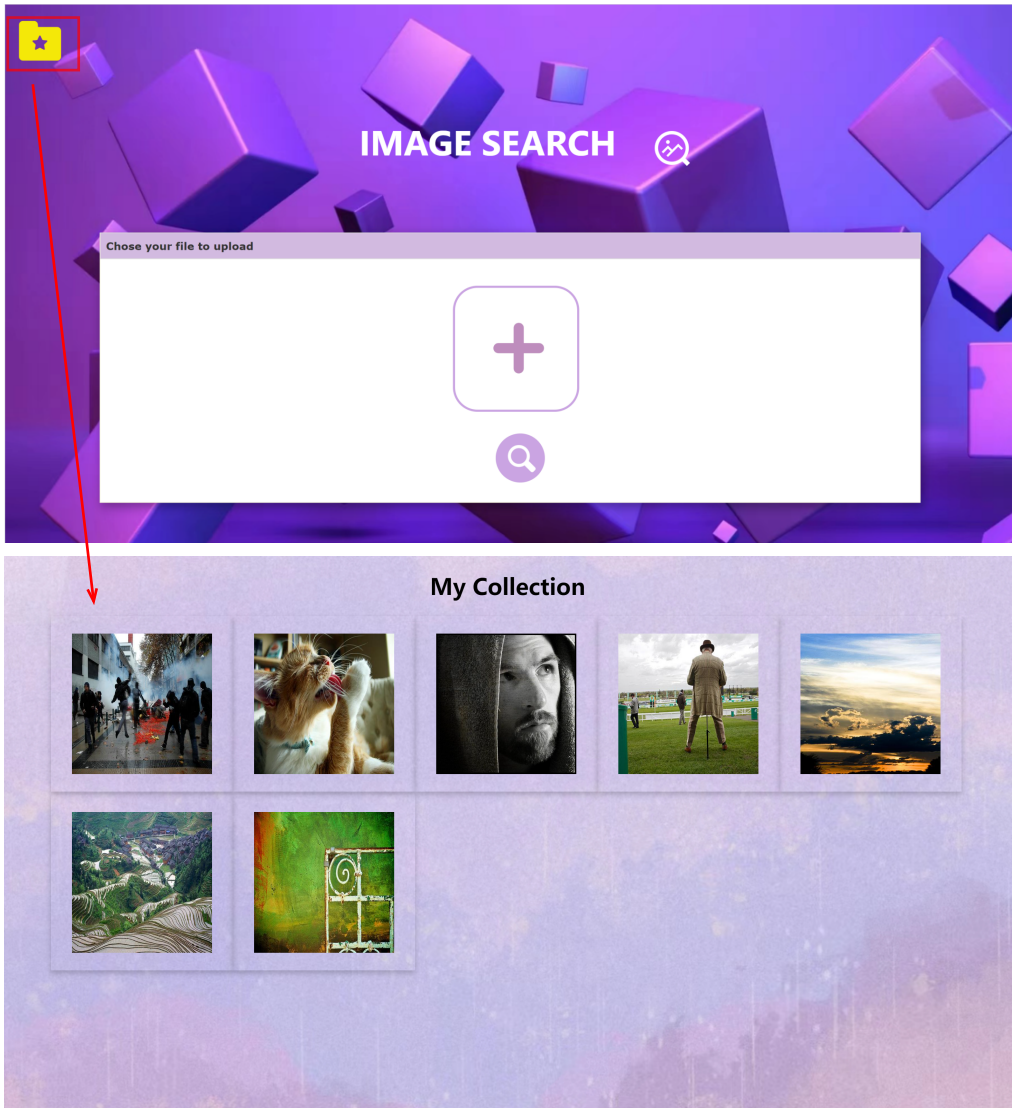
    for example: The user selects the tag `structures` and filters out the images that contain the tag in the results.



- **Use**
  - Users can add selected images to a collection list
    - Users can click the stars to add a picture into collection list, and click it again to delete the picture from the collection list.



    - Users can click on the favorite icon located in the top left corner to view their collection of saved images.



Implement

- ## Image upload and preview

```
<div class="select" >
    <input id="file_input" type="file" name="file" onchange="FileUpload()"/>
    <img id="selectImg" src="" >
</div>
```

```
 // 显示上传的图片
function FileUpload()
{
    var fileInput = document.getElementById('file_input');
    var uploadedImage = document.getElementById('selectImg');

    fileInput.addEventListener('change', (event) => {
        var file = event.target.files[0];
        var reader = new FileReader();


        reader.readAsDataURL(file);

        reader.onload = () => {
            uploadedImage.src = reader.result;
```

```
            uploadedImage.style.display = 'block';
        }
    });
}
```

- ## Search for results

The retrieval request is sent to the back end through AJAX.

`response.images` is the image retrieval result passed in by the back end

```
$.ajax({
    url: 'imgUpload',
    type: 'POST',
    data: formData,
    cache: false,
    contentType: false,
    enctype: 'multipart/form-data',
    processData: false,

    success: function (response) {
        $('#load').hide();
        $('#row1').show();
        // 遍历reponse中的图片数量
        for(i=0;i<9;i++){
            $('#img' + i).parent().show();
        }
        document.getElementById("img0").src = response.images.image0;
        document.getElementById("img1").src = response.images.image1;
        document.getElementById("img2").src = response.images.image2;
        document.getElementById("img3").src = response.images.image3;
        document.getElementById("img4").src = response.images.image4;
        document.getElementById("img5").src = response.images.image5;
        document.getElementById("img6").src = response.images.image6;
        document.getElementById("img7").src = response.images.image7;
        document.getElementById("img8").src = response.images.image8;
        document.getElementById('result_count').innerHTML = "为您找到结果共9条"
})
```

- ## Review of result

`div` holds the tag corresponding to the uploaded image

```
<div class="tag"></div>
```

Create tag elements dynamically:

```
const labelsContainer = document.querySelector('.tag');
response.upload_tag.forEach(tag => {
    const span = document.createElement('span');
    span.textContent = tag;
    span.classList.add('label');
    labelsContainer.appendChild(span);
});
```

`response.upload_tag` is an array of tags corresponding to the image uploaded by the user through the back end.

```
@app.route('/imgUpload',methods=['GET','POST'])
def upload_img():
     #the other codes...
     # Gets the name of the uploaded file
     uploadfile = file.filename
     # Extract the number in the file name
     upload_num = uploadfile.split('.')[0][-1]
     # Empty the upload_tag array
     upload_tag=[]
     # Check whether the number appears in the tag list
     for i in tags:
         if upload_num in tagDict[i]:
             upload_tag.append(i)
    #the other codes...
    upload_dict={"images":images,"upload_tag":upload_tag,"img_tag":img_tag}
    return jsonify(upload_dict)
```

- ## Change search parameters

Add a listener for each tag element, and when the user clicks on a tag, send a filter request to the back end through AJAX. The `response` is the filter result passed in from the back end, hiding the filtered image based on the filter result.

```
// Gets all the elements of the class named label
var checkboxes = document.querySelectorAll('span.label');
// Add a listener for the tag element
for(var i=0;i<checkboxes.length;i++){
    checkboxes[i].addEventListener("click",function(event){
        var selected=event.target;
        for(var j=0;j<checkboxes.length;j++){
            checkboxes[j].style.backgroundColor='rgb(154, 171, 219)';
        }
        selected.style.backgroundColor='#caa4e2';
        select_tag=selected.textContent;
        $.ajax({
            url: 'filter',
            type: 'POST',
            data:{
                select_tag:select_tag
            },
            success: function (response) {
                console.log("filting success!");
                console.log(response);

                // Iterate over the number of images in reponse
                for(i=0;i<9;i++){
                    $('#img' + i).parent().show();
                }
                // Display filtering results
                for (i = 0; i < response.num; i++) {
                    document.getElementById('img' + i).src =
response.result[i];
                }
```

```
                // The filtering result is not displayed
                for (j = response.num; j < 9; j++) {
                    // document.getElementById('img' +
j).parentNode.style.display = 'none';
                    $('#img' + j).parent().hide();
                }
                //  Update overview
                document.getElementById('result_count').innerHTML = "为您找到
结果共"+response.num+"条"
            }
        })
    })
}
```

Back-end `filter` function:

```
@app.route('/filter', methods=['POST'])
def filter_images():
    select_tag = request.form.get('select_tag')
    result=[]
    #Total number of screening results
    count=0
    print("select_tag",select_tag)
    for i in tagDict[select_tag]:
        result_path='static/result/im' + i + '.jpg'  # the URL of the
filtered image
        if os.path.exists(result_path):
            print(i)
            result_path = '/result/im' + i + '.jpg'
            result.append(result_path)
            count=count+1
    print(count)
    response={'num':count,'result':result}

    return jsonify(response)
```

- ## Add/Delete collection

the button of "collect":

```
<button class="collect" data-img="" ></button>
```

Each button element is traversed and a click event listener is added to it. Once the user clicks the `.collect` button below a certain image, the collected/uncollected request is sent to the back-end through AJAX. The `response` is the request result returned by the back-end, and the corresponding button state is modified according to the result:

```
// Gets all button elements whose class is collect
var btnList = document.getElementsByClassName("collect");
//
for (var i = 0; i < btnList.length; i++) {
    var btn=btnList[i];
    var imgSrc = btn.previousElementSibling.src;

    // Add listener
```

```
    btnList[i].addEventListener("click", function(event){
        var button=event.target;
        var imgSrc = button.previousElementSibling.src;
        $.ajax({
            url:"collect",
            type:"POST",
            data:{
                imgSrc:imgSrc
            },
            success: function(response) {
                //Favorites before clicking button
                if(response.img_collect){
                    button.style.backgroundImage="url(images/collect-
icon.png)";
                }
                else{  //No favorites before clicking button
                    button.style.backgroundImage="url(images/collected-
icon.png)";
                }
            },
            error: function(xhr) {
                console.log(xhr);
            }
        });
    });
}
```

Back-end `add_collection` function:

```
@app.route('/collect',methods=['POST','GET'])
def add_collection():
    imgSrc = request.form.get('imgSrc')

    #get name of the image
    new_filename = os.path.basename(imgSrc)
    #modify imgSrc
    imgSrc = os.path.join('static','collection', new_filename)
    #Whether the selected image file name is in the Favorites collection, if
so, img_collect=true,
    img_collect = os.path.exists(imgSrc);
    print(img_collect)
    print(imgSrc)


    # image have been collected
    if(img_collect):
        os.remove(imgSrc) # delete
    # image not collected
    else:
        # create collection folder
        folder = 'static/collection'
        if not gfile.Exists(folder):
            os.mkdir(folder)
        # Copy the image to the static folder
        with open('database/dataset/' + new_filename, mode='rb') as f:
            byte_data = f.read()
        with open('static/collection/' + new_filename, 'wb') as f:
```

```
            f.write(byte_data)

    return jsonify({'img_collect':img_collect})
```

- ## View collections

The favorites flag in the top left corner of `main.html` ,Clicking this flag will take you to `collection.html`

```
<a class="mycollection" href= "{{ url_for('mycollection') }}">
        <img src="/images/collection.png" alt="" width="60px">
</a>
```

`collection.html` is rendered via a back-end route

```
@app.route('/mycollection')
def mycollection():
    return render_template('collection.html')
```

```
 <!--colleciton.html-->
 <div class="img-container">
```

Get all the links of the user's favorite pictures and display them through a back end request with AJAX:

```
$(document).ready(function() {
    $.ajax({
        url: '/show_collection',
        type: 'POST',
        success: function(data) {
            const Container = document.querySelector('.img-container');
            data.img_paths.forEach(path => {
                const img = document.createElement('img');
                //Some image css style Settings
                Container.appendChild(img);
            });
        },
    });
});
```

Back-end `show_collection` function:

```
@app.route('/show_collection',methods=['POST','GET'])
def show_collection():
    collection_path = 'static/collection'
    # img_paths = [os.path.join(collection_path, img_name) for img_name in
os.listdir(collection_path)]

    collection = "/collection"
    img_paths = [os.path.join(collection, file) for file in
os.listdir(collection_path)
                 if not file.startswith('.')]

    return jsonify({'img_paths':img_paths})
```