Lab2: Image Retrieval

> @author 1851055 Mingjie Wang

Installation

- To run the code, you should install `conda` environment like the following:

  ```
  pip install -r requirements.txt
  ```

  On the other hand, you should also download and unzip **database.zip** in `server`.

  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
    |  |  favorites.txt
    |  |
    |  ├─dataset
    |  |
```

```
    │   └─tags
    │
    ├─frontend
    │
    ├─imagenet
    │       classify_image_graph_def.pb
    │
    ├─static
    │   ├─images
    │   │
    │   └─result
    │
    ├─template
    │
    └─uploads
```

Task Requirements

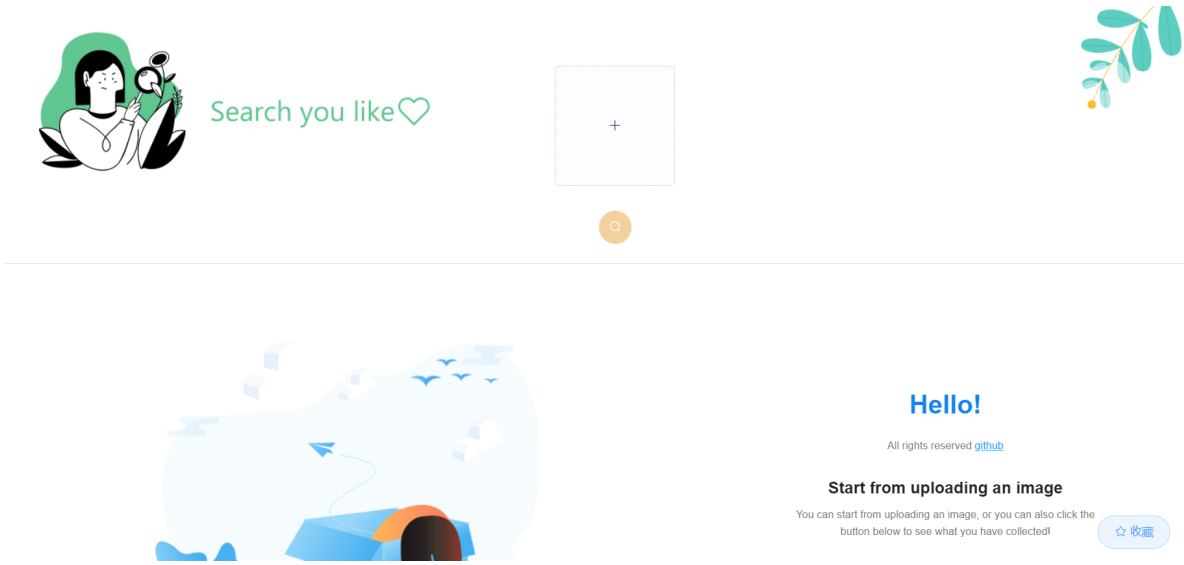
The requirements of an image search task is as follows:

1. **Formulation**:
    - It contains an input box to upload an image.
    - Users can preview the query image in the searching window.
2. **Initiation**: It has a search button.
3. **Preview**: Provide an overview of the results.
4. **Refinement**: Allow changing search parameters when reviewing results.
5. **Use**: Users can take some actions, like add selected images to collection.

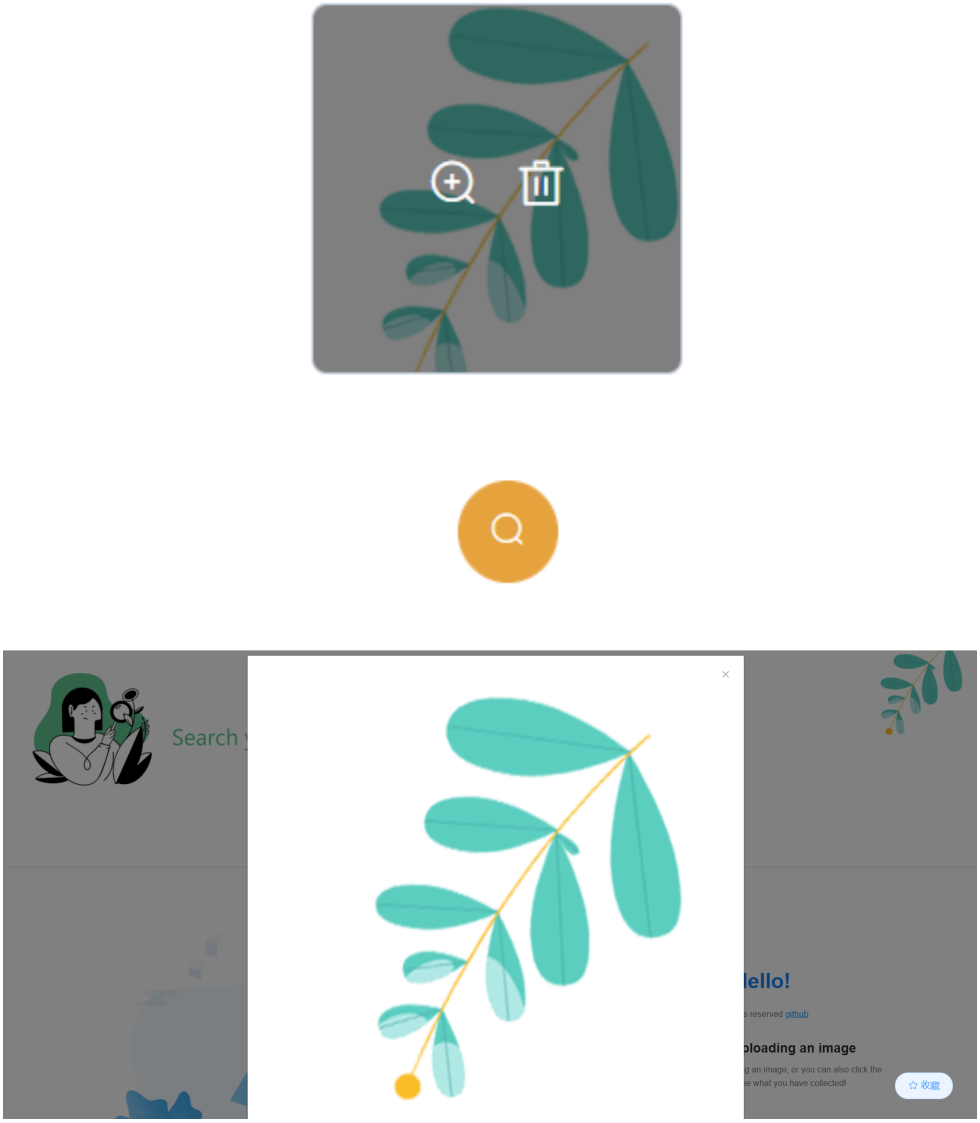
Functionality

Formulation

- Upload the image



- Preview the image uploaded

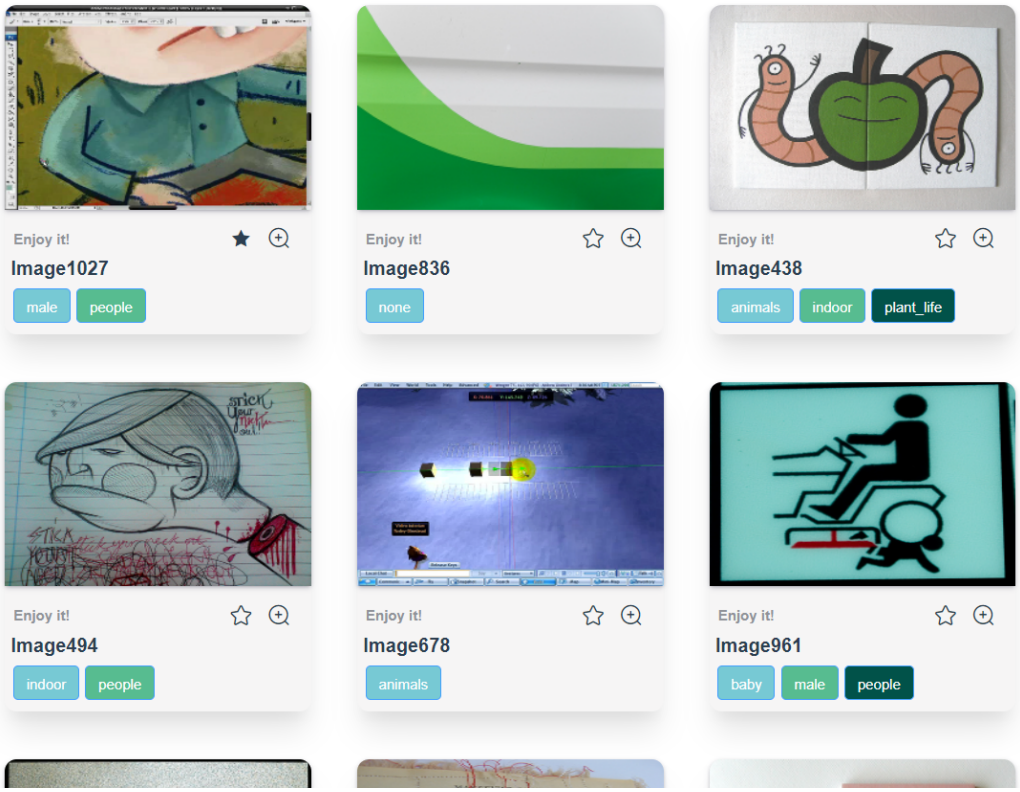


Initiation

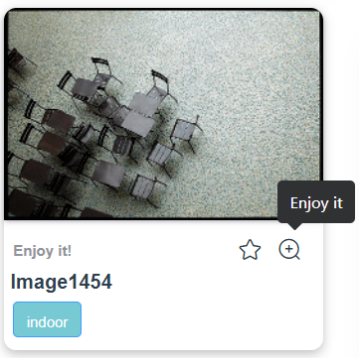
Click the search button for result:

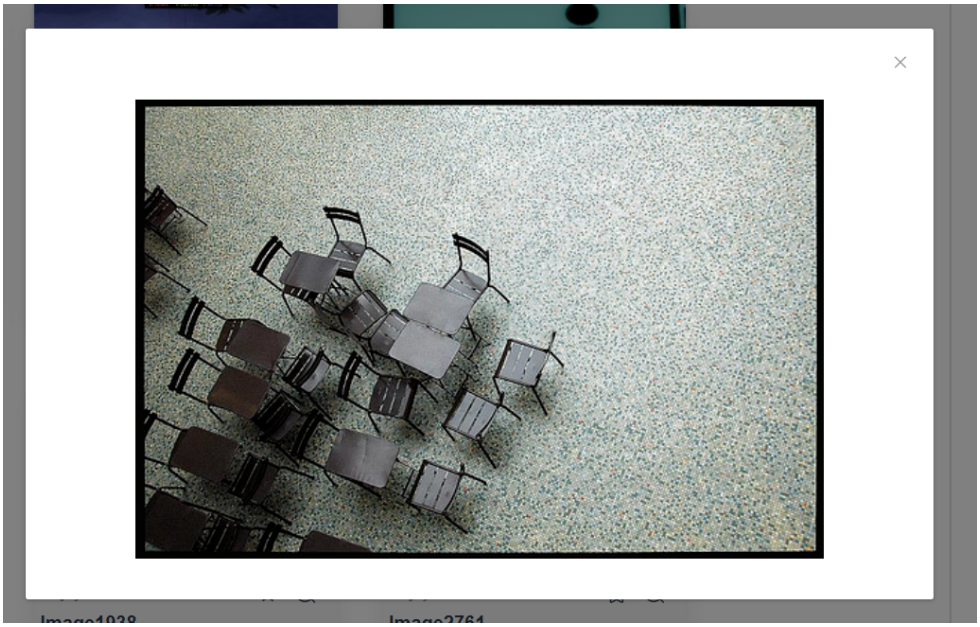
Preview:

- The overview of the results:



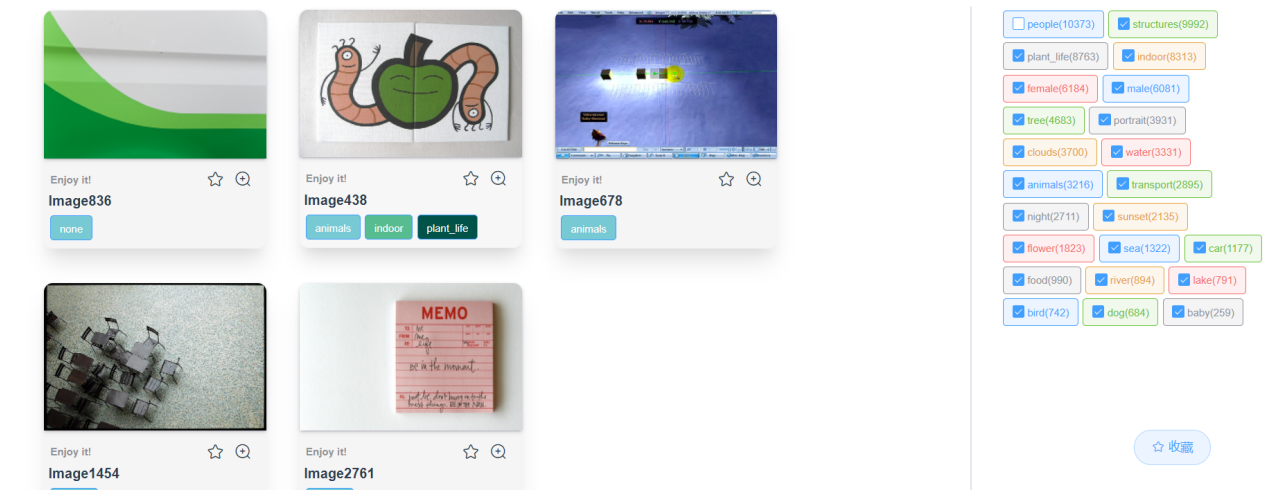
- Enjoy the result image:





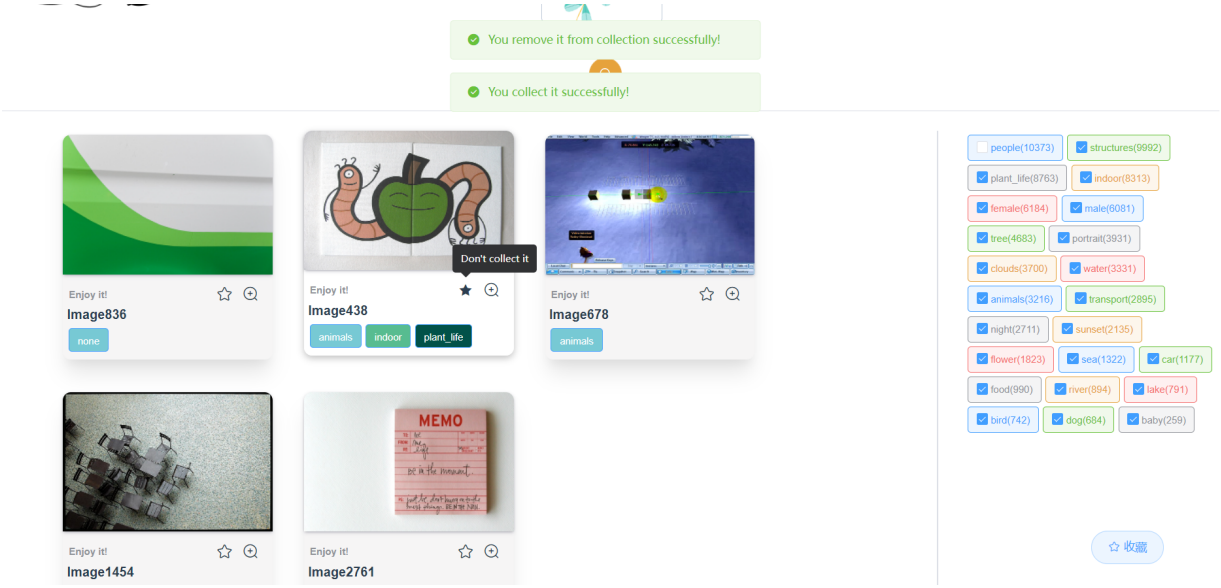
Refinement

Change searching parameters(image tags on the right):

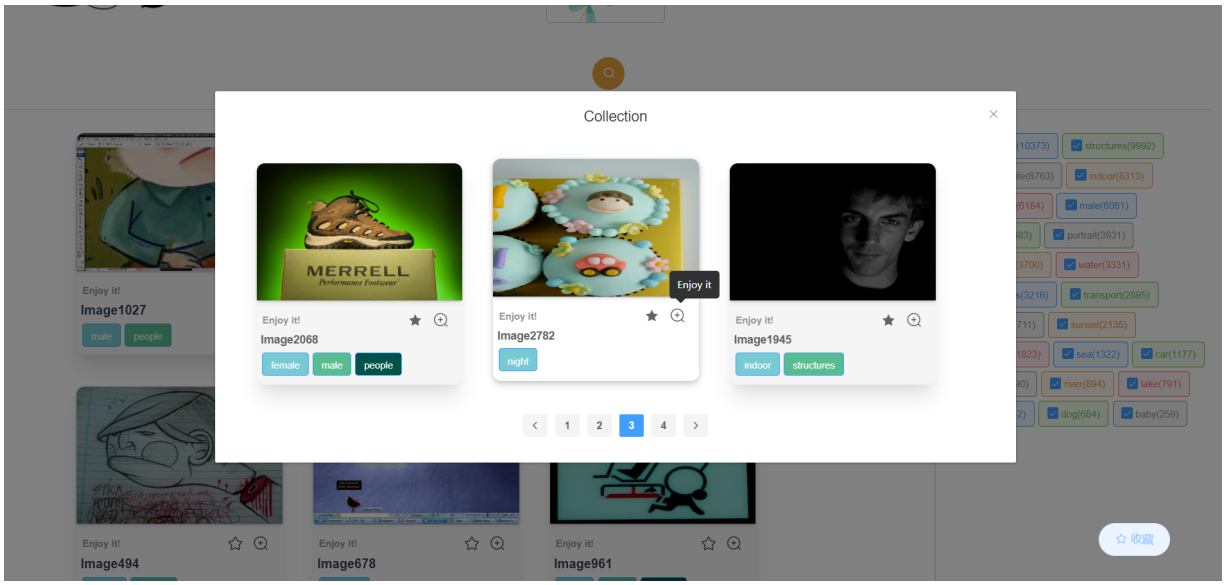


Use:

- Collect an image:



- See collection list:



Implement

In this project, I use `Vue` as the interface development framework.
After the development is completed, it is packaged into index by `npm run build` command.
And the source file is placed in the `/template` folder.

In the end, call `index.html` like this in the `flask`:

```
@app.route('/')
def main():
    return render_template('index.html', name='index')  # 使用模板插件，引入index.html
```

Of course, you can directly run the interface code in the `/frontend` folder:

```
cd frontend
```

```
npm install
npm run serve
```

Then you can visit the interface: http://localhost:8080/

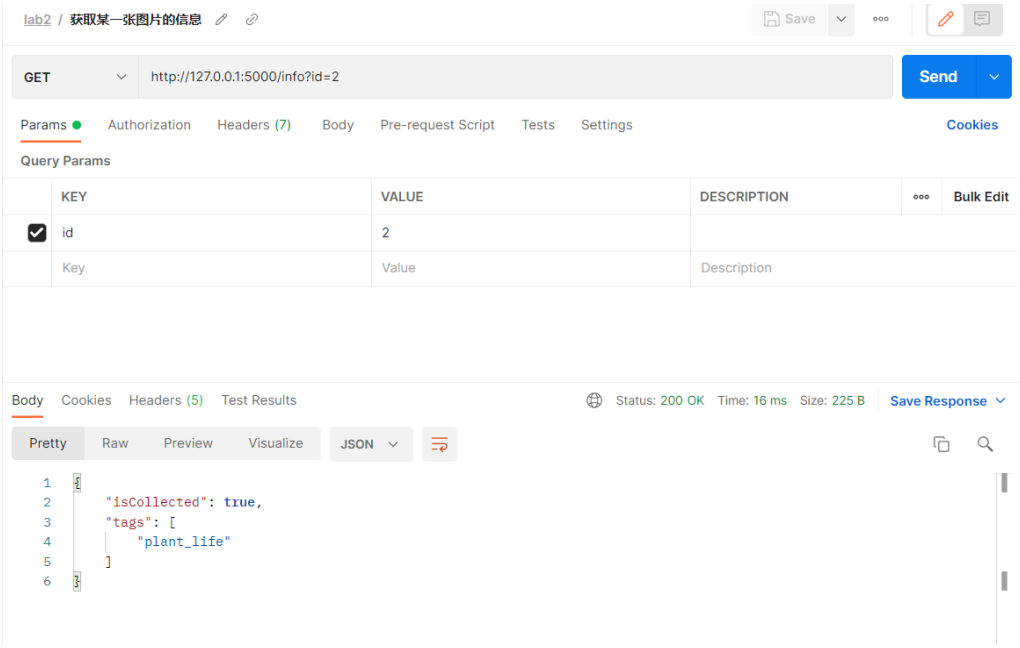
Upload

For uploading an image, I use the component implemented in `Element-UI`. The following file comes from `/src/components/UploadImage.vue`:

```
<template>
    <div>
        <el-upload
        action="imgUpload"
        :limit="1"
        list-type="picture-card"
        :auto-upload="false"
        :multiple="false"
        :file-list="fileList"
        :on-success="uploadSuccess"
        :on-change="handleChange"
        accept="image/png, image/jpeg"
        ref="uploadRef"
        :class="fileList.length >=1 ? 'styleB' : 'styleA'"
        >
            <em slot="default" class="el-icon-plus"></em>
            <div slot="file" slot-scope="{file}">
                <img class="el-upload-list__item-thumbnail" :src="file.url" alt="">
                <span class="el-upload-list__item-actions">
                    <span class="el-upload-list__item-preview"
@click="handlePictureCardPreview(file)">
                        <em class="el-icon-zoom-in"></em>
                    </span>
                    <span v-if="!disabled" class="el-upload-list__item-delete"
@click="handleRemove(file)">
                        <em class="el-icon-delete"></em>
                    </span>
                </span>
            </div>
        </el-upload>
        <el-dialog :visible.sync="dialogVisible">
            <img width="80%" height="80%" :src="dialogImageUrl" alt="">
        </el-dialog>
    </div>
</template>
<script>
    export default {
```

```
        data() {
            return {
                dialogImageUrl: '',
                dialogVisible: false,
                disabled: false,
            };
        },
        props:{
            fileList: Array,
        },
        methods: {
            handleRemove() {
                this.fileList.splice(0, 1);
            },
            handlePictureCardPreview(file) {
                this.dialogImageUrl = file.url;
                this.dialogVisible = true;
            },
            uploadSuccess(response) {
                this.$emit('uploadSuccess', response);
            },
            handleChange(file){
                if (this.fileList.length == 1) {
                    return;
                }
                this.fileList.push(file);
            },
            submitUpload(){
                this.$refs.uploadRef.submit();
            },
        }
    }
</script>
```

Image Card



In order to have a better visual experience, I separately encapsulate the image card in a `Vue` file:

```
<template>
    <div>
        <div v-if="isTagDisallowed" class="CardContainer">
            <div class="CardType" @mouseenter="changeCardStyle($event)"
@mouseleave="removeCardStyle($event)">
                <!---->
                <el-image fit="fill"
                    class="card-image"
                    :src="'image?id='+imageId">
                    <div slot="error" class="image-slot">
                        <img fit="fill"
                            class="card-image"
                            src="@/assets/404_images/error.png" />
                    </div>
                </el-image>
                <el-row style="margin-top: 10px;">
                    <el-col :span="17">
                        <h5
                            class="imple-text">
                            Enjoy it!
                        </h5>
                    </el-col>
                    <el-col :span="3">
                        <!--收藏图片-->
                        <el-tooltip class="item" effect="dark"
:content="isCollectedLoading? 'waiting...' :
                    (isCollected ? 'Don\'t collect it': 'Collect it')
                    " placement="top-start">
                            <div v-if="!isCollectedLoading">
                                <em :class="isCollected? 'el-icon-star-on':
```

```
                            'el-icon-star-off'" style="font-size: 20px;"
@click="collectImage"></em>
                            </div>
                            <div v-else>
                                <em class="el-icon-loading" style="font-size: 20px"></em>
                            </div>
                        </el-tooltip>
                    </el-col>
                    <el-col :span="3">
                        <!--查看大图-->
                        <el-tooltip class="item" effect="dark" content="Enjoy it"
placement="top-start">
                            <em class="el-icon-zoom-in" style="font-size: 20px;"
@click="viewBigImage"></em>
                        </el-tooltip>
                    </el-col>
                </el-row>
                <h4 class="main-text">
                    Image{{imageId}}
                </h4>
                <div class="label-list">
                    <el-tag type="primary" v-for="(i,index) in showTags" :key="index"
effect="dark"
                        :color="labelColor[index]" :hit="true">
                        {{i}}
                    </el-tag>
                </div>

                <el-dialog :visible.sync="dialogVisible">
                    <img width="80%" height="80%" :src="'image?id='+imageId" alt="">
                </el-dialog>

            </div>
        </div>
    </div>

</template>
```

API

The rest API is used in `flask`, and `postman` is used as the testing tool.

```
{
    "name": "获取某一张图片",
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "http://127.0.0.1:5000/image?id=2",
            "query": [
```

```
                {
                    "key": "id",
                    "value": "2"
                }
            ]
        }
    },
    "response": []
},
{
    "name": "获取某一张图片的信息",
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "http://127.0.0.1:5000/info?id=2",
            "query": [
                {
                    "key": "id",
                    "value": "2"
                }
            ]
        }
    },
    "response": []
},
{
    "name": "收藏/取消收藏图片",
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "http://127.0.0.1:5000/collect?id=2",
            ]
        }
    },
    "response": []
},
{
    "name": "获取全部标签",
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "http://127.0.0.1:5000/tags",
        }
    },
    "response": []
},
{
    "name": "获取用户全部收藏",
```

```
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "http://127.0.0.1:5000/collect/all",
        }
    },
    "response": []
}
```