Mini Project ::

Installation steps::

Download Zip file React-Django-Project

Install and activate the virtual environment in both the terminals using ::

pip install pipenv

pipenv shell

change directory to React-Django-Project in both the terminals using cd React-Django-Project

Again change directory to React-Django-Project both the terminals cd React-Django-Project

Backend terminal:: Install all the requiremnts using below command::

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

then run python manage.py runserver

(Backend is set up)

If we want to check it then can go to browser and type: (These are the api's that are created)

http://127.0.0.1:8000/api/image-list/

http://127.0.0.1:8000/api/tag-delete/1/ (instead of 1 can type any tag id that we wish to delete and that exists)

http://127.0.0.1:8000/api/image-tag-add/1/

Frontend terminal::

Change directory

cd Frontend

Run command ::

(if npm is not installed in the laptop then first Install Node.js and Create a Local Development Environment)

npm install

Run command ::

npm start

(Our frontend is also set up)

Now we can go to :: http://localhost:3000/ and should see the following as shown in the video attached ::

https://drive.google.com/file/d/1fg_4SV4R1cxZSy4SoTRcc5lXmY5V_Wdv/view

How I have implemented Django + React :::

Backend::

Created Models Image and Tag in models.py(Django) in sqlite3 database.

Tag → field :: tag_name

Image → fields :: Image id primary key

```
            description

            img = models.ImageField(upload_to="images/")
            x

            y

            height

            width

            tags = models.ManyToManyField(Tag)
```

Here I have linked class Image and Tag using manytomanyfield .

Now I have created 3 api's using Django rest framework which provides serializers(that converts our data into JSON format) and using @api_view(['GET']), @api_view(['DELETE']), @api_view(['POST'])

The 3 total api's are

```
1)imageList of type @api_view(['GET']) → will give list of images and also give details of How many tags are linked to each image.
I have used CustomSerializer for giving information(details) about tag that is tag name.


2)tagDelete(request, pk) of type @api_view(['DELETE']) → To delete a tag(We will have tag id and will delete that particular tag)


3)imageTagAdd(request, pk): of type @api_view(['POST']) → To add a tag(we will get image id  where we wish to  add  new tag + new tag name and  I have created new tag and add that in our image and save its object at backend)
```

and one serializer that is

1)ImageCustomSerializer

where I am returning all the information about the image i.e image id , description, imagelink , height , width , x and y etc. and Taglist(which is cutomfield) where I have passed image id + image link as a List.

Now Frontend ::

State :: where we 3 main fields that is images[] → The list of images[alongwith the information of linked tags] that will come from from ImageList Api will be stored here.

tag_names →This is used for The feature of adding/deleting tags according to image_id.{and the currently used tags(the ones which would be displayed)}

show_tag_detail_of → show_tag_detail_of - it will decide which image-container will have tag-container

```
constructor(props) {
        super(props);
        this.state = {
            images: [],
            tag_names: {},
            show_tag_detail_of: 1,
        }
```

Then Here I am fetching the information about the images i. e data into images using my api http://127.0.0.1:8000/api/image-list/ and stored in images(my state).

```
    fetchImages() {
        console.log('Fetching...')

        fetch('http://127.0.0.1:8000/api/image-list/')
            .then(response => response.json())
            .then(data =>
                this.setState({
                    images: data
                })
            )
    }
```

Similarly fetched for delete tag api and again reload it by calling this.fetchimages().

```
   deleteTag(tag) {
        var csrftoken = this.getCookie('csrftoken')

        fetch(`http://127.0.0.1:8000/api/tag-delete/${tag.id}/`, {
            method: 'DELETE',
            headers: {
                'Content-type': 'application/json',
                'X-CSRFToken': csrftoken,
            },
        }).then((response) => {
            this.fetchImages()
        })
    }
```

addTag →Here I am fetching the image add tag api I .e http://127.0.0.1:8000/api/image-tag-add/1/

.Here I have used tag_name unlike deleteTag where I used tag_id as we need to add new tag.

And after adding new tag we will reload using this.fetchImages() , at last clear the values that are already added in tag_input.

```
addTag(e, image_obj) {
        e.preventDefault();

        var image_id = image_obj.image_id;
        var csrftoken = this.getCookie('csrftoken')
        var url = `http://127.0.0.1:8000/api/image-tag-add/${image_id}/`;

        var data = {
            tag_name : this.state.tag_names[image_id]
        }

        fetch(url, {
            method: 'POST',
            headers: {
                'Content-type': 'application/json',
                'X-CSRFToken': csrftoken,
            },
            body: JSON.stringify(data)
        }).then((response) => {
            this.fetchImages()
            var tag_inputs = document.getElementsByClassName("tag_input_elements"
);
            for (var i=0; i<tag_inputs.length; i++) {
```

```
                tag_inputs[i].value = "";
            }
        }).catch(function(error) {
            console.log('ERROR:', error)
        })

    }
```

handleTagChange :: After adding the new tag name(event) that name would be added to its corresponding image_id.

```
  handleTagChange(e, image_obj) {
        e.preventDefault()

        var image_id = image_obj.image_id;
        var new_tag_name = e.target.value;

        this.state.tag_names[image_id] = new_tag_name;
    }
```

Render()

Basic structure :: Main container

- Every image will get one image-container
- Then in which image-container , tag-container is to be shown would be decided by show_tag_detail_of.
- Then in tag-container there are two things → one to display the tags(display tag-name and the button to delete tag name) and other to add the tags[using forms(input div and submit button division)]

```
main-div
    image-container
        image-div
            img
        tag-container
            tag-add-div
                form
                    form-div
                        input-div
                        submit-button-div
            tag-list-div
                div
                    tag-name-div
                    tag-delete-div
```

```
    image-container
        image-div
            img

    image-container
        image-div
            img
```

```
onClick={(e) => self.changeTagDetailId(e, image_obj)  → To go on which division id it is clicked on (So if we will click on first image then that id would be changed to corresponding tag id)
```

For deleteTag ::

```
<button onClick={() => self.deleteTag(tag_obj)} className="btn btn-sm btn-outline-dark delete">-</button>
```

For adding new tag ::

```
 <input onChange={(e) => self.handleTagChange(e, image_obj)} className="form-control tag_input_elements" type="text" name="tag_name" placeholder="Tag Name" />
```

Bounding Box:: I think of the approach of doing bounding box but was not able to do it , my approach is to store the measurements in my state , fetch the box on mount using [const box = this.text.getBBox() and update its dimensions in the state using box.width and box.height and then render the updated state.

Summary ::

When the product is opened on broswser , images from the database are loaded and it allows the user to click on the image of his/her choice , when the user clicks on the image , it loads the tags from the server stored till now(displays the tags) , and also allows the user to add/remove one or more tags.

My product is able to use react and save it to the django server, and load the tags from the server at a later time ie if browser is closed then the previously added tags would also be shown when the user opens the browser in the later time.

Thank you.