

**PIXEL PROJECT**

**Prepared for**

StJohn Russel

VP Engineering, TooMuchWifi

**By**

Mpetha S. Motaung

Software Engineer, TooMuchWifi

14.12.2024

# Abstract

# Introduction

Before you get started here are the prerequisites:

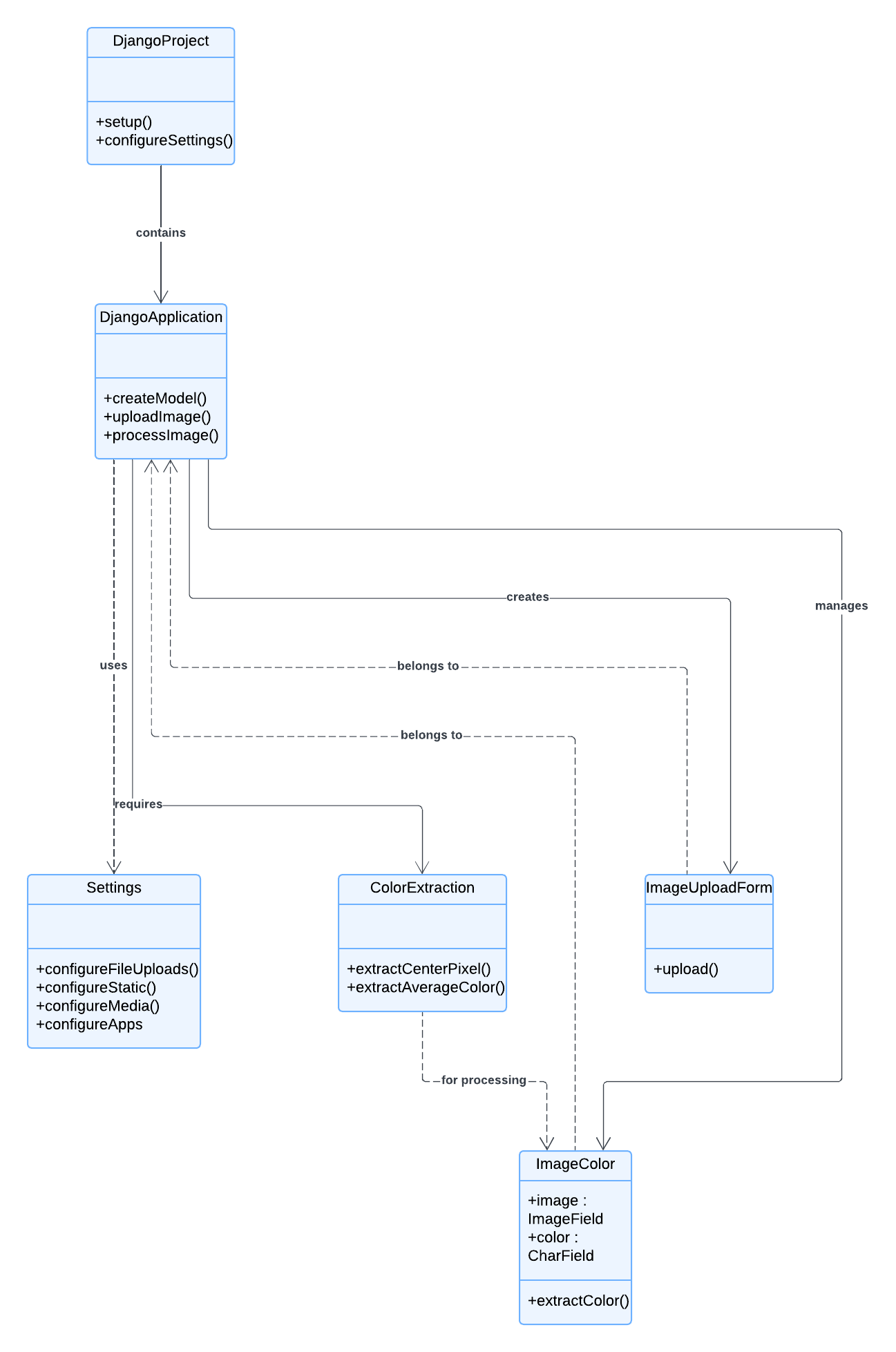
* Understand virtual environments
* Understand package managers
* Understand cmd commands
* Understand Github Desktop, Github and Gitversion control

## Objectives (Initial Assumptions)

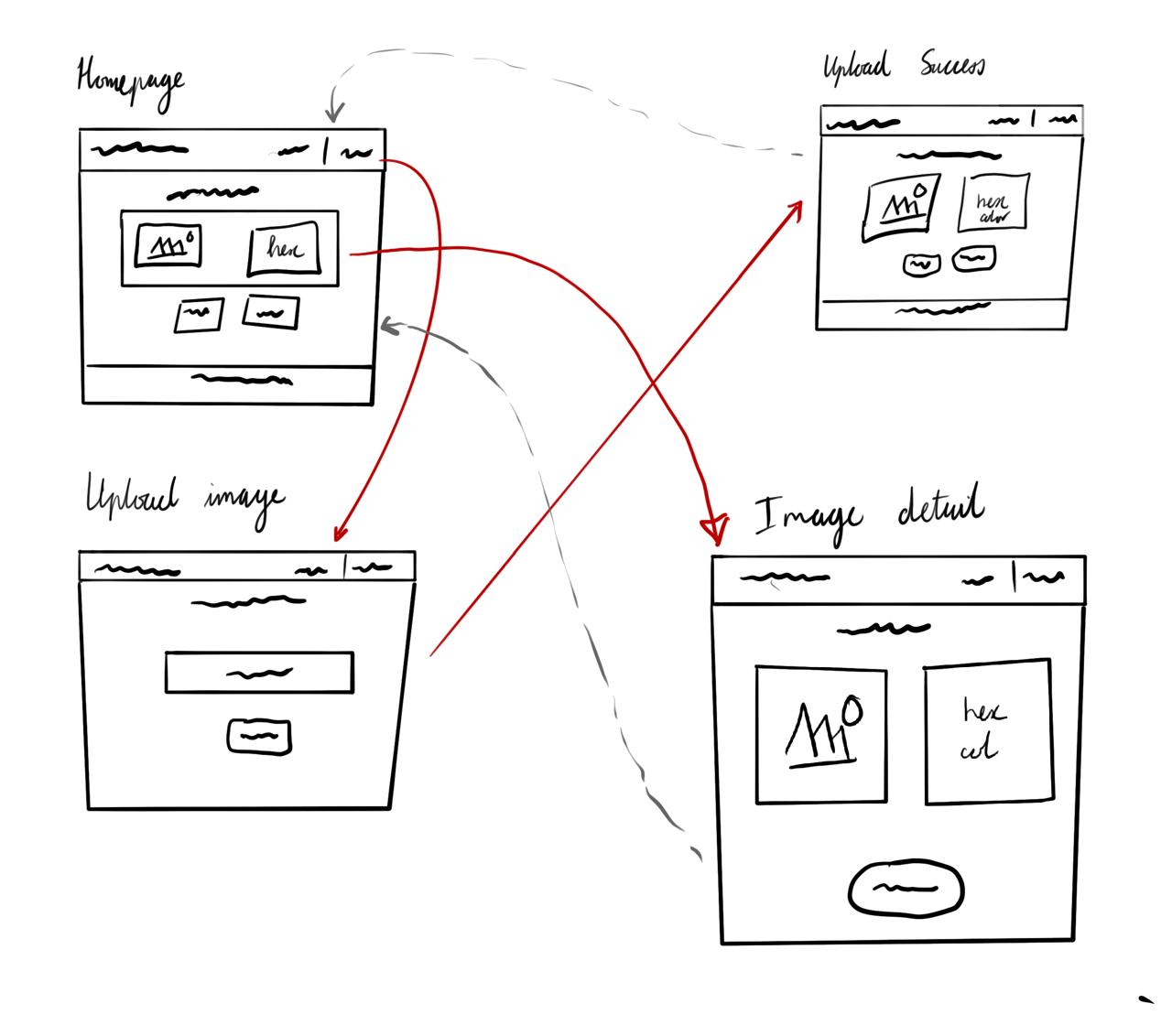
* Create Django project that can handle image uploads.
* The uploaded images have to be processed to extract center most pixel and hex value.
* Both uploaded image and hex value pair have to be viewable by the user (As a gallery of photos and their hex values).
* The uploaded images and their hex values must be displayed using Django’s template system.
* The web application must be able to handle errors for non-image file uploads and image processing fail cases.
* Minimum memory usage when extracting center pixels without loading the entire image (Save the url of the image as text in the database.)

## Architecture

### Overview



### Design Decisions



##### Page Design/Layout

###### Homepage (image\_list [index)

###### Upload Image page (upload\_image.html)

###### Upload Success page (upload\_success.html)

###### Image Details page (image\_details.html)

## Workflows

# Setup & Project Configuration

Download and install Anaconda for package management and virtual environment. (used for sourcing, building, and deploying data science and AI initiatives)

\*You may use Anaconda or any other virtual environment platform (e.g. virtual env, pyenv, pipenv) but I prefer Anaconda\*

After Downloading <https://www.anaconda.com/download/success> and installing Anaconda. There are 2 methods of creating a virtual env

### Creating or choosing Project location

You can either create a folder using the Windows Graphic User Interface (GUI) or Command Line Interface (CLI). We must create a folder anywhere in file explorer but I suggest using Document/Github/(project name) to ensure that all your projects are organized. [[1](#_References)]

#### Method 1 (Windows GUI)

1. Open File Explorer (double left-click)



1. Navigate to documents, create a Github folder(if it doesn’t already exist). Within Github folder create project folder ‘Image Processing’ (Documents/Github/Image-Processing)



This is the folder that we will be executing the Django project from

#### Method 2 (CMD)

1. Open CMD, then navigate to Documents/Github. Create the Github folder if it doesn’t already exist.



To navigate into documents cd documents, then hit enter

1. Navigate into github folder



To navigate into Github folder in documents cd Github, then hit enter

1. Create project folder for our Django project



This will create a folder for our Django project but it is still not connected to Github or any repository on Github

1. Next open Github desktop, create a new repository and link it to the folder we just created





OR

Open Github Desktop, Go to File> Add local repository



Point Github to the local repository/folder that you want to add to Github Desktop. Then publish to Github.



#### Alternatively (To save time)

\*Note\* You must have Github Desktop installed for you to proceed with this alternative method [[2](#_References)]

1. Open Github Desktop and create new repository. Github Desktop will automatically create a project folder on your local machine that will be used through and IDE to interact with Github.



1. Give your repository a name, this will also automatically be the name of the folder that Github will create in your chosen path. Read up on Github Desktop, how to create and publish a repository [[3](#_References)]



Now that our folder has been created and we have created a new repository and linked it to our folder we can move to the next step

### Downloading and Installing Anaconda

1. Click on the Download button



1. You’ll be redirected to this page and your download should start



1. Once the file has finished downloading (/downloads), double click on the file and install Anaconda













1. Now that you have downloaded and installed Anaconda. Let’s launch it for the first time

\*If you have never used Anaconda, I suggest you watch a few short tutorials (YouTube or on the Anaconda website). It’s quite user friendly\*



### Creating a virtual environment - Method 1 (GUI)

1. Open Anaconda
2. On the sidebar on the left click>environments
3. Then click create

* Name your environment according to the project that you’re working on so that you can remember it. In this case I will name this one ‘Pixels’



1. Anaconda will now create a virtual environment with the python version and the name that you chose. It should take 2-5 minutes



1. Next you can hit the play button to check your virtual environment. If it is created successfully, you should see the following:



The name of our virtual environment inside the brackets confirm that our virtual environment was created successfully and is running successfully.

1. Type pip list in the terminal to see all the installed packages in the Pixels virtual environment. Since we do not have any additionally installed. We will only see the default packages that are built in(default) with every environment creation using pip



### Creating a virtual environment - Method 2 (CMD)

1. You can follow along with CommandPromt(CMD) or open the terminal window in an IDE of your choice. In this case, I’m going to use my preferred IDE, which is VSCODE



1. Navigate to the terminal and type the following command conda create –name (name of your environment) python=3.12



You can choose any name for your environment but to keep things simple and aligned, go with Pixels. Additionally, you can choose any python version to install on your virtual environment (i.e, python=3.8, python=2.7). Knowing this will be vital so that you can test your software in different versions of python to ensure consistency and to test for issues.

1. Conda will now create a virtual environment





Choose ‘y’ to proceed to install the suggested packages in your environment.

1. Now that we have created our virtual environment let’s try to activate it and test if it has been successfully created. Run the following command conda activate Pixels



If you followed all the steps correctly, you should see (Pixels) which is showing that your virtual environment has been created successfully.

1. When you run the command pip list, you should see the default installed packages when a virtual env is initiated through Anaconda



### Install Django

Now let’s install Django into our virtual environment. Run the following command pip install Django (\*note\*: if you do not specify the Django version, pip will install the latest Django version)



\*Ensure that when working on your project, that your virtual environment is active before running any commands or trying to install packages. Remember we do not want to modify our system packages thus we created a virtual environment. \*

If the above command runs successfully, you should see the following



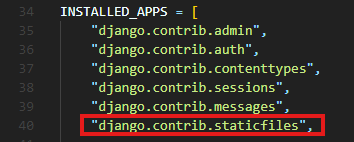
## Install Django Debug Toolbar

This tool is useful for debugging Django web applications. The toolbar helps you understand how your django application functions and to identify problems [ ]

1. Run pip install django-debug-toolbar

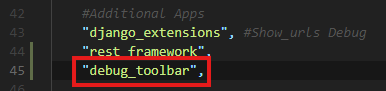


1. Ensure that django.contrib.staticfiles is in installed\_apps in settings pixelsproject/pixelsproject/settings.py

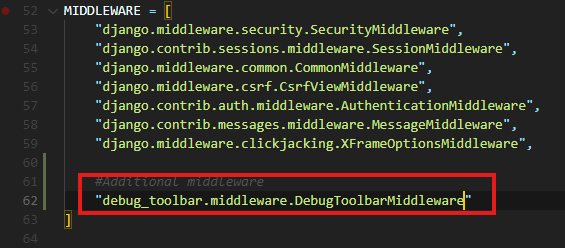




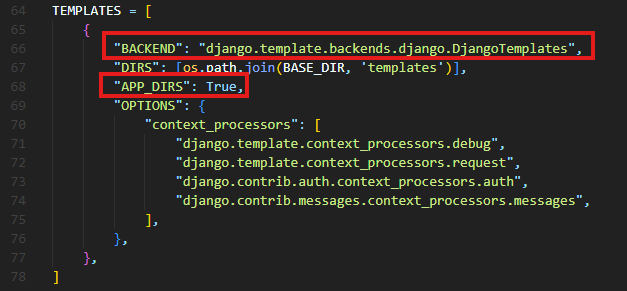
1. Add the app debug\_toolbar to installed\_apps



1. Add debug\_toolbar middleware



1. Ensure TEMPLATES section in settings contains DjangoTemplates backend with APP\_DIRS set to true



### Create Django Project

Now before we dive into it, let’s make sure that you’re in the correct directory. In my case I’m in Visual Studio code CMD terminal



#### Create Django project [[4](#_References)]

Run command django-admin startproject (myprojectname)



After you run the command a project folder will be created with the name that you selected when you ran the command django-admin startproject (myprojectname)

#### Now let’s create an app to handle the processing of our images. Navigate into the Pixelsproject



#### In the pixelsproject path, run the following command to create Django app

\*Before you run the following command, check if you are in the correct directory by running this command first dir



This is our main Django project folder with the manage.py and Pixelsproject folder

#### After ensuring that you’re in the correct path. Run the following command python manage.py startapp pixels to create our pixels app [[5](#_References)]



You should now see a folder named pixels



#### Registering pixels app in settings.py of main PixelsProject

Navigate to PixelsProject>Settings.Py

In Settings.py under installed apps, add pixels app



\*Note\* in your IDE make sure that you have selected your virtual environment to avoid errors



#### Create urls.py file for pixels and link to main urls [[4](#_References)]

Create urls.py file for app pixels’ urls



Link app pixels urls to main urls.py of Pixelsproject []



### Create & Configure static folder and files

This is to serve additional files such as images, javascript, or css [ ]

1. Let’s create a static folder in our app {{{{{{{ conda install anaconda-navigator

**Option 1 (create through IDE GUI)**



**Option 2 (create through CLI)**

* Navigate to pixels app



* Run the command mkdir static



* You should now see a new folder in the pixels app named ‘static’

\*Best practice is also to create static sub folders for future usage\*

1. Navigate to pixelsproject/pixelsproject/settings.py []

First, we want to import os [] for directory handling and processing



Add directory paths in the Static files section of settings.py [ ]



Configure templates in settings.py



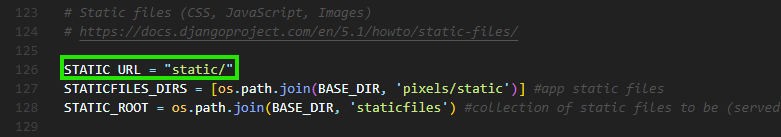
Configure Media (directory configuration) []



### File management static files:

Websites have to serve additional files such as image, JavaScript, or CSS. In Django they are referred to as static files. Django provides django.contrib.staticfiles to help manage them. [ ]

1. Ensure that django.contrib.staticfiles is included in installed\_apps section in project settings.py



1. In templates, use the static template tag to build URL for given path using static files STORAGES

Add to all template files to load static files



1. Create static folder to store static files

File structure: pixels/static/pixels/[css,js,images]



# Model Creation

## Open models.py of pixels app (pixelsproject/pixels/models.py) and define a class with Imagefield and charfield



Next, go to the CLI to make migrations. After creating a model in Django, before running our server (the server can still run without making migrations but you will get a makemigrations error) we have to makemigrations to create a migrations file that contains code for the tabled schema of our model. This is done through the Django ORM. It creates a table according to the schema defined in the migration file.

1. Run command python manage.py makemigrations

Encountered errors



Static file settings incorrect and pillow not installed (Pillow package is a prerequisite for using ImageField) []

1. Run command pip install pillow



1. Run command manage.py makemigrations to initialize our database



Because we have configured settings for global static(Pixelproject/PixelProject/Settings.py). We need to create a global static folder



In our PixelsProject root folder, let’s create static folder to resolve this issue:



1. Now that we have installed pillow and created a global static folder. Let’s run migrations again. Because we have already ran migrations. We now have a migrations file:



This is the initial migrations file

So now when we run migrations. We will see the following:



This is because we have already ran migrations to our database but have not made any modifications to our model class.

1. Run python manage.py migrate to apply changes to our database



# Image Upload & Processing

1. Let’s create a form.py for handling user image uploads [] []

Create forms.py file in Pixels app (PixelsProject/pixels/forms.py)



1. Now let’s create utility for processing images through Pillow



# Views Creation

## Overview



1. First, we have to import necessary dependencies and the created py files (forms, models, utils)



1. Secondly, we need functions to handle image uploads, image list, and image detail

**Upload image function**



**List image function**

****

**Image detail function**

****

# URLS creation(app)

## Overview



1. Firstly, let’s import the necessary dependencies and views



1. Define app\_namespace so that app urls are not added to global namespace. If app\_name = ‘yourappname’ is not defined it will lead to url conflicts at a later stage. [ ]



1. Then we define urls patterns to route the functions that we created in our views



# HTML Templates Creation

## Introduction

The django template system is designed for allowing ease of use with HTML experience. The system is meant to express presentation, not programming logic. It provides tags which function similarly to some programming constructs. []

### Base.html template



### Upload.html template



### Image\_list.html template



### Image\_detail template



# Implementing Atomic Transactions

This is to resolve the issue of running SQL injections because Django has an auto-commit function that commits SQL injections without verifying all statements in a view function and not undoing the commit if a statement fails. So we have to add Atomic transaction to overwrite Django’s auto-commit SQL to avoid having our database modified but we are still getting errors. [ ]

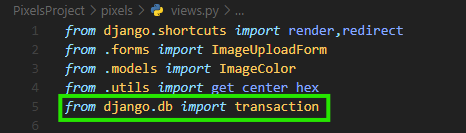
## Atomic transactions have the following properties:

* Atomicity: Ensures all transactions succeed or none does.
* Consistency: Ensures data consistency
* Isolation: Transactions (SQL query) happen in isolation
* Durability: committed transactions remain in the system even if the system crashes

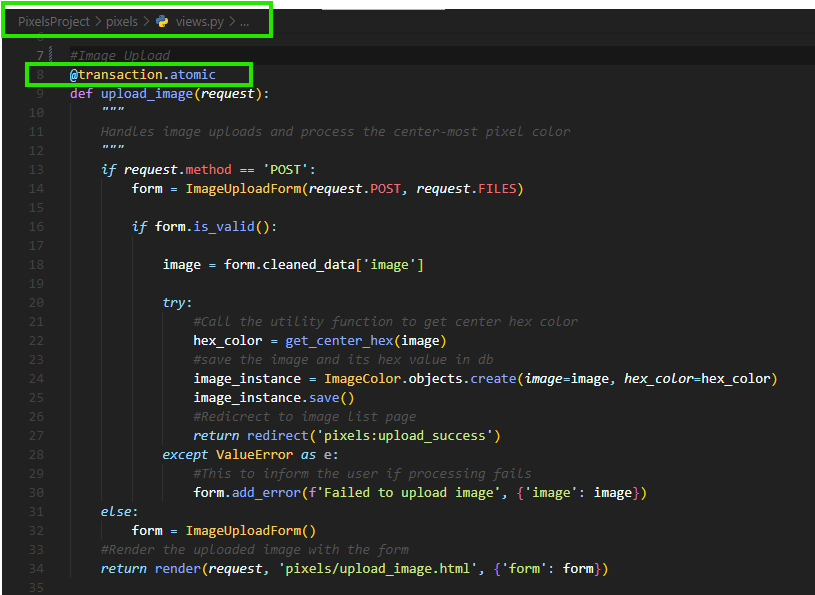
## Advantages of using atomic transactions in Django:

* Data consistency: data base remains the same even if system crashes and only commits changes only when operations in a transaction are successful
* Error Handling: rolling back transactions if an error occurs
* Enhance code readability: marking transactional sections
* Improved database performance: minimizes database commits, which can lead to enhanced database performance

First let’s import transactions in views



Then wrap any functions that write to the db in @transaction.atomic



# Testing the server

Encountered errors:

## Error 1:

## Error 2:





### Solution:

Missed the s in image\_list function



Add pxiels folder in templates to fix the directory path. The correct path is pixels/templates/pixels/



Solution change import structure and use single quotation ‘ ’ for app url path []



## Error 3:





Either create global templates folder for base/layout template or move base.hmtl from pixels/templates/pixels/ to pixels/templates



## Error 4:





### Solution:

Add {% load static %} to base.html



## Error 5:







### Solution:

Forgot to close the tag correct code is



## Error 6:



### Solution:



## Error 7:







### Solution:

Add pixel app name to template {% url ‘app\_name:url\_name’ %}



## Error 8:





### Solution

Fix pixelsproject/pixels/utils.py for loop. 4 channels is used for RGBA. Which does not align with the purpose of our code. Since we are converting our image to RGB. We must loop through 3 channels only  


Call created util function to get hex color





Result, when user clicks upload image



## Error 9:





### Solution

It seem that Python is not recognizing our image\_list. This error might mean that we have to fix our pixels app urls for the exit or there is a syntax(typo) in our urls

**This is our pixels app urls file**



The first path path('', views.image\_list, *name*='image\_list'), is our focus since the error NoReverseMatch is refering to the reverse match for image\_list. As this is correct django will be able to redirect(‘image\_list’) [ ]

**Here is our image list view function.**

Which looks correct. Created an image\_list function that calls all image color objects and then returns the image\_list page.



**Let’s try the image\_list.html**

we had the same problem in base.html and image detail and solved it by adding our app name to url call. Incorrect {% url ‘image\_list’ %} Correct: {% url ‘pixels:image\_list‘ %}



## Eror 10:



Reverse for ‘image\_detail’ not found. ‘image\_details’ is not a valid view function or pattern name.



This is another url problem

### Solution

Check urls, views and templates for syntax errors or spelling errors



Since we defined our render request to serve image\_details.html We have to use this same name for referring to this file in the creation of our templates

Incorrect:  Correct

Now, we’ve checked views and template file name

**Let’s check our app urls**

****

In the url path for image details, we correctly defined the path and assigned a primary key. Imported image\_detail from views.py and name.

**Let’s check our templates where we reference the image\_details.html**

We do not reference/call image\_details.html in our base.html file, yet. So, we do not need to check it

We do not reference/call image\_details.html in our upload.html file, yet. So, we do not need to check it

We have to check image\_list.html because that is the source of our error

****

Here is the current image\_list.html

****

Our focus is line 9 in image\_list.html, where we route an anchor link that redirects to image\_detail.html

Let’s modify the href from {% url ‘pixels:image\_detail’ image.id %} to {% url ‘pixels:image\_details’ image.id %} because the template we are trying to access has a name of image\_details that we just modified above.



Run python manage.py runserver to refresh the server and check if our error is fixed. At this point we don’t have to clear the cache because we do not have css styling yet.

Fixed

## Repeated error:

We are getting the error of NoReverseMatch on image\_list view function when trying to upload an image. Let’s debug and find out why we are getting this error.



Let’s traceback and diagnose again. We should’ve used Django shell to save ourselves the headache of running into the same error again.

### Solution:

Let’s double check our urls.py file in pixels app pixels/urls.py

**Current state**



All seems to correct

Let’s check our main project urls.py file pixelsproject/urls.py

**Current State**



All looks correct

**Let’s double-check our views for the image\_list function**

We’ve run import django.shortcuts import render and from .models import ImageColor and defined our fuction for image\_list

Current state



All looks correct.

**Let’s check templates for {% url %} usage [] []**

Let’s rename our upload template from upload.html to upload\_image.html so that Django can find it because our path is currently being routed to a template that does not exist.

Here’s the url.py of our app pixels/urls.py that is routing the Django search path for upload image to ‘upload\_image’



Incorrect  correct 

Now, let’s double check the upload\_image.html



We didn’t include a way for navigating back to the image list after an image is uploaded. Let’s add that



**Now, let’s use Django’s shell debug to verify our fix before running our server again**

Run python manage.py show\_urls [ ] to confirm URL patterns.

\*Note, you have to install django\_extensions first\* []



This is the output. Confirm image\_list exists:



**Now, let’s run our server to check if error has been fixed**

Run python manage.py runserver



Proceed by uploading and image

## Error 11





### Solution

In our views.py at pxiels/views.py we defined the upload\_image function but our return render request path is incorrect because it is referencing a template that does not exist

Incorrect:



Correct, according to our change of upload.html to upload\_image.html:



## Error 12:



### Solution

In views.py pixelsproject/pixels/views.py in upload\_image function we forgot to update the render redirect path from return redirect(‘image\_list’) to return redirect(‘pixels:image\_list’)

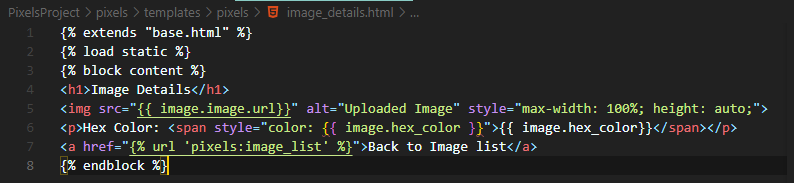
Incorrect:



Correct:



## Error 13:



At line 6, We want to render the hex value to display the actual color. Might have to use some css to achieve this. This is a technique in Django templates, to dynamically set styles based on data passed to the template [ ] We might have to use a template engine to generate html dynamically so that we can access and manipulate the parsed data.

Solution:

Error 14:

From python server



Solution:

Let’s modify the path of styles static according to your file structure pixelsproject/pixels/static/pixels/css/styles.css



Error 15:

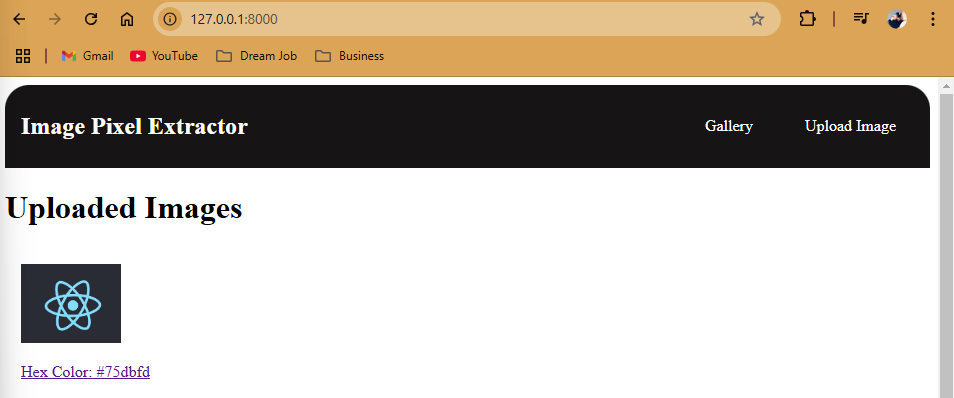
Django SQL auto commit is causing images to be uploaded even after the flow fails. So we have to validate all transactions before committing changes to database.

Solution:  
Implement Atomic transactions with ACID properties [ ]

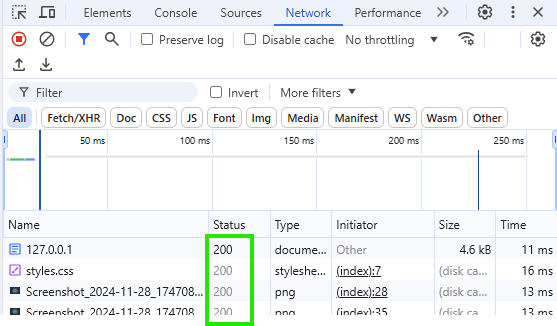
## Test Case 1

|  |  |
| --- | --- |
| Purpose: Test if Django is correctly setup along with application and that there are no errors when navigating through the entire app. | |
| **Test** | **Expected results:** |
| Django app(pixels) navigation | Navigate through root template(image\_list) and other templates with no errors.   * Server running * Root template served correctly * Css styles served correctly * Response **200** |

**Root (‘ ‘) template(index/homepage), served by our dev server:**

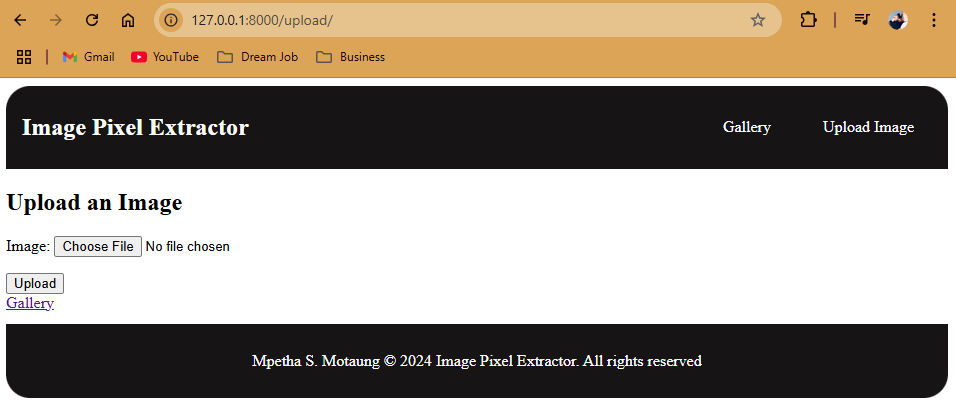


Developer tools > Network (Check response/status)

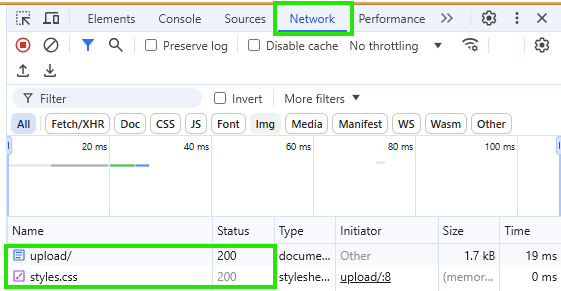


**Upload image template(upload\_image.html), served by our dev server.**

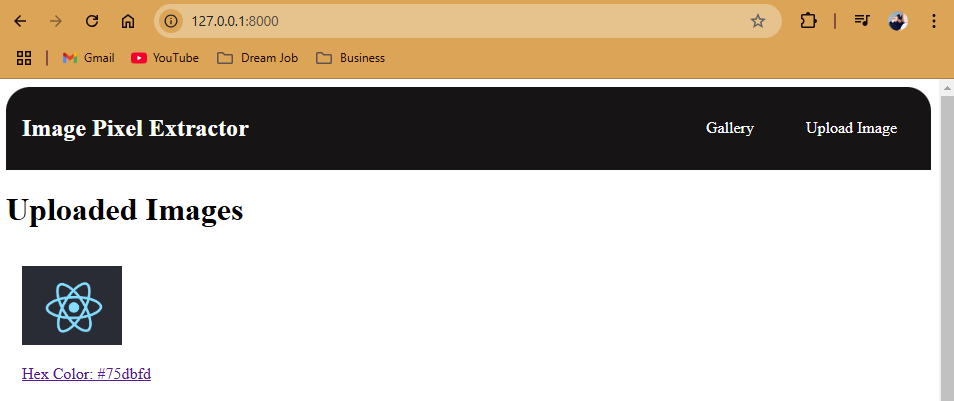
From (‘ ’) root > click Upload Image nav link :



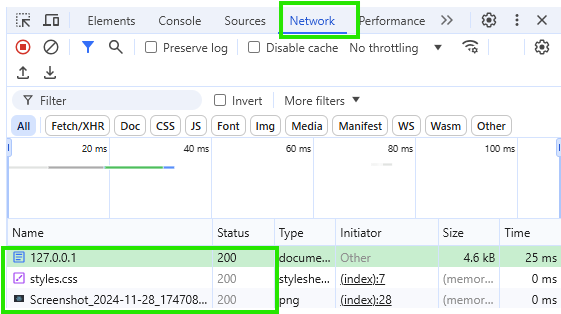
Developer tools > Network (check results after ‘upload image’ in the navbar is clicked



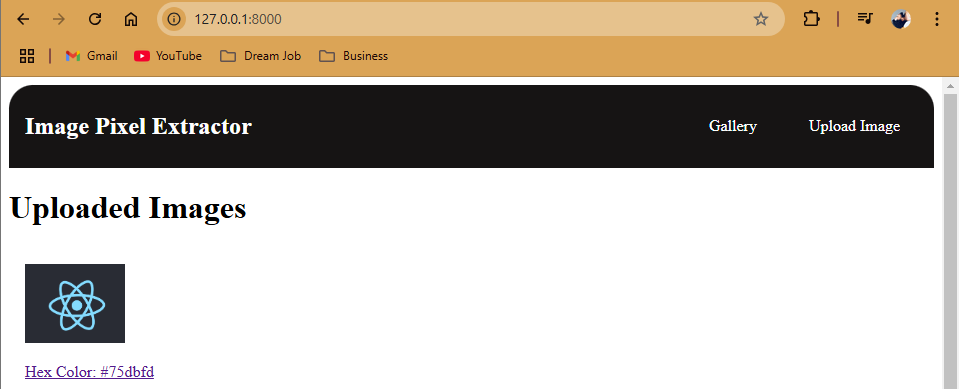
**From (‘upload’) > root ( ‘ ‘ ) gallery using Gallery button:**



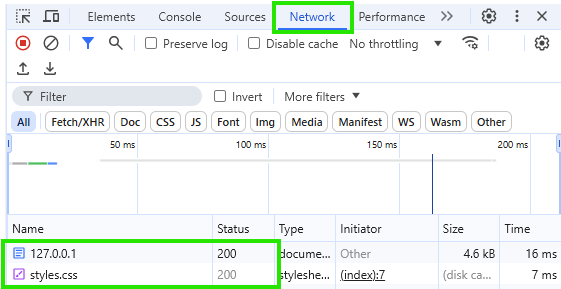
Developer tools > Network (check results when ‘Gallery’ in the navbar is clicked



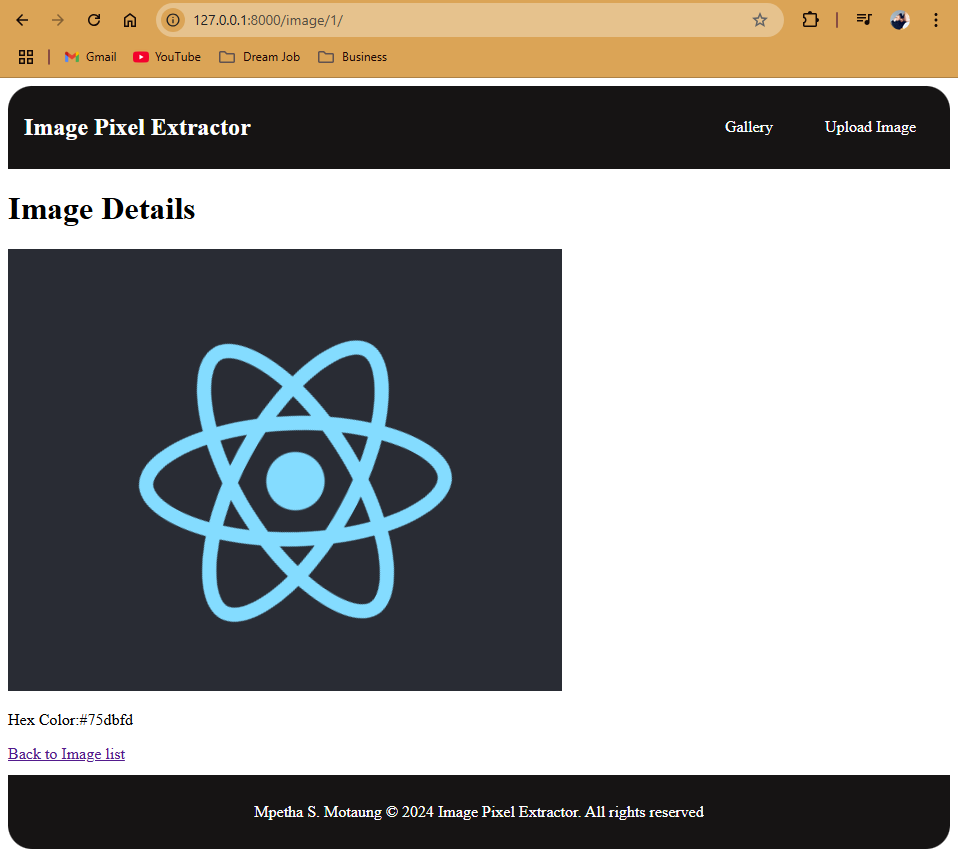
**From (‘upload’) > root ( ‘ ‘ ) gallery using ’Gallery’ in the navbar:**

****

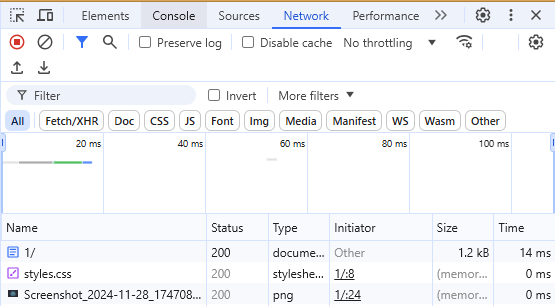
Developer tools > Network (check results when ‘Gallery’ in the navbar is clicked

****

**Image details template(image\_details), accessed through clicking on the picture or clicking on the hex color**



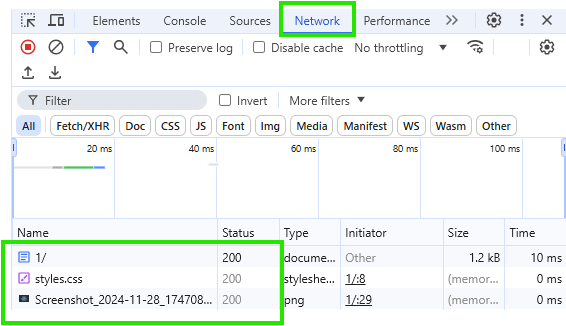
Dev tools > Network (check response when image\_details.html is accessed)



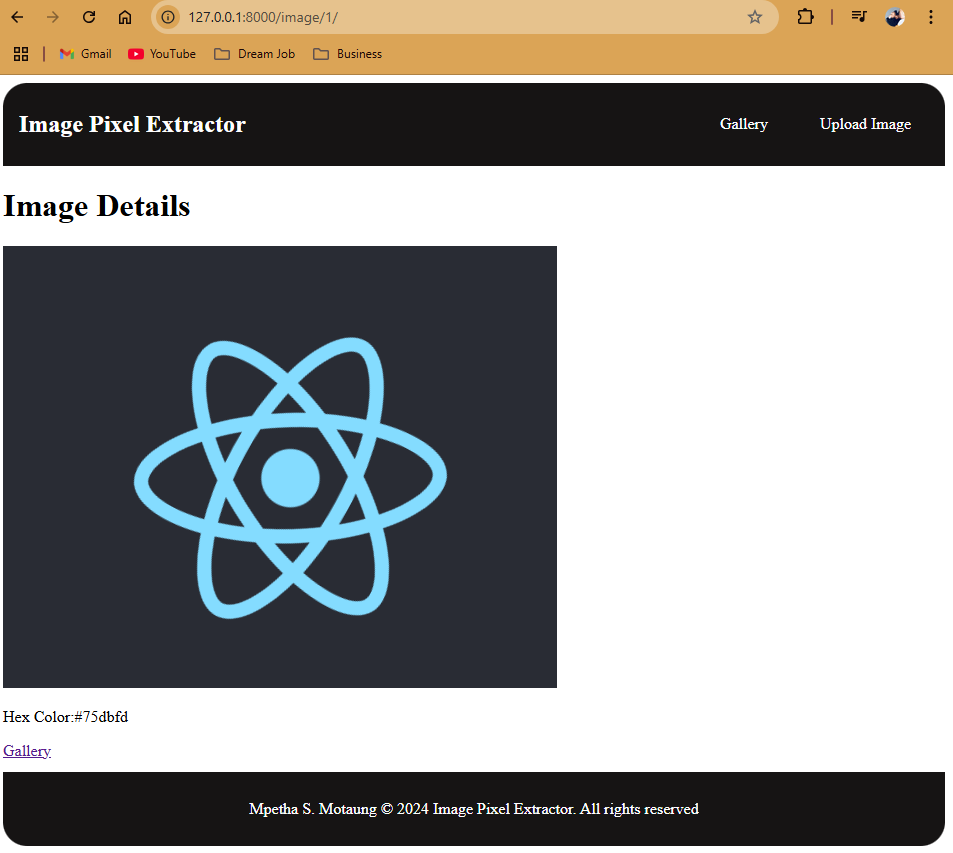
**From root (‘ ‘) image\_list > view image details by clicking on the image:**

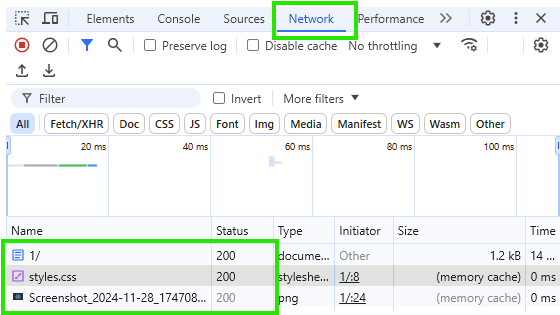


Dev tools> network (check response when image is clicked to access image details)



**From root (‘ ’) image\_list > view image details by clicking on image hex color**

-



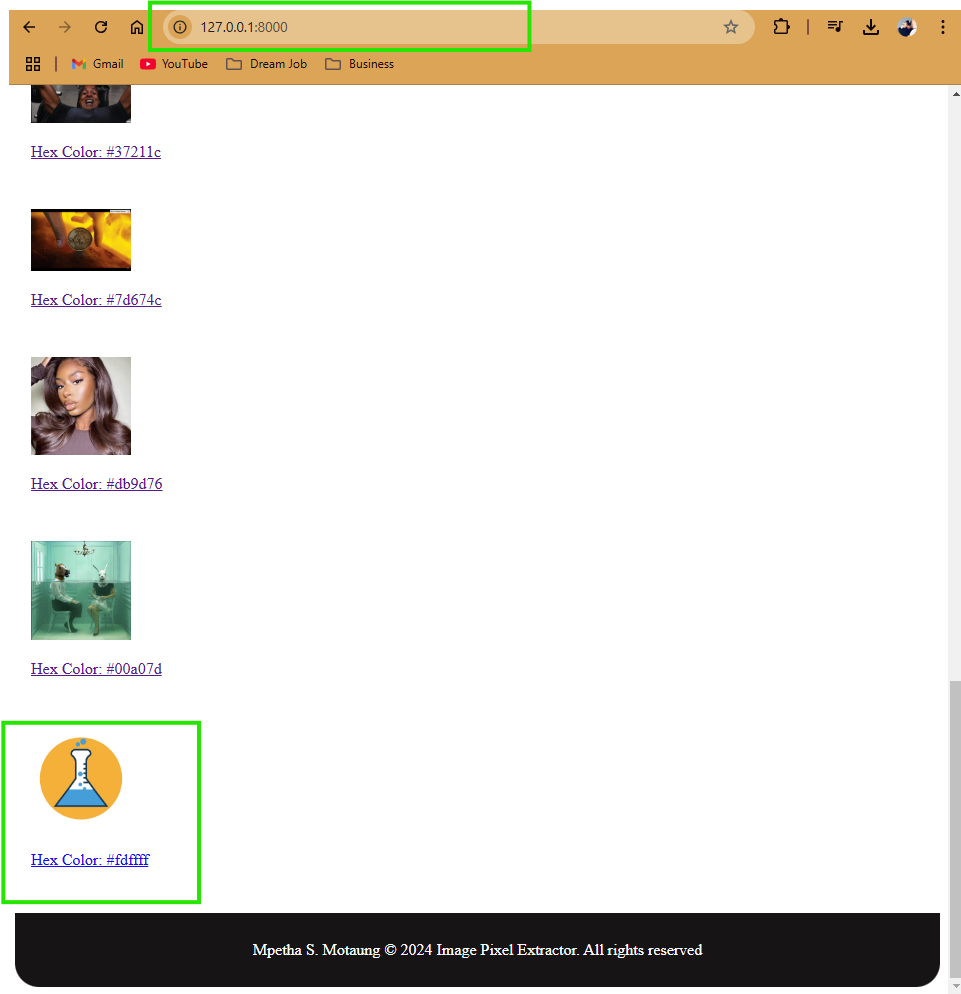
## Test Case 2

|  |  |
| --- | --- |
| Purpose: Test whether file uploads are handled correctly through file upload configuration. | |
| **Test** | **Expected results:** |
| Upload File | * Uploaded image through form is parsed through algorithm and rendered correctly   + Uploaded image is processed and served in image\_list through our configuration. |

Choose image file > upload

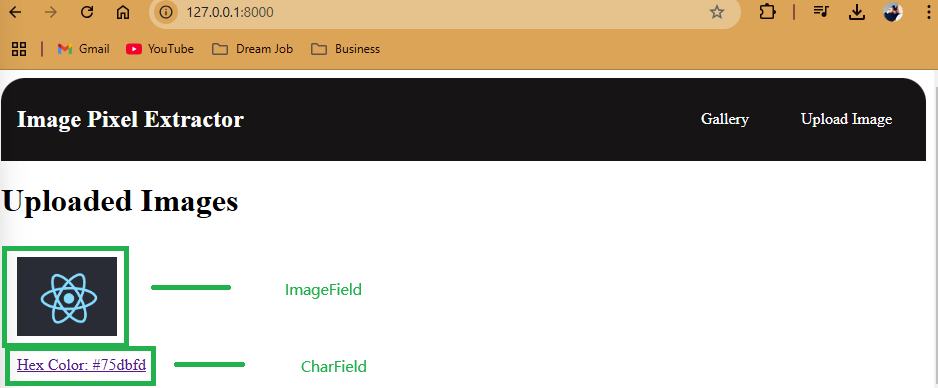


Navigate back to gallery to check if image has been processed and center-most pixel is extracted. The image, along with its hex color should be displayed on our image list:



## Test Case 3

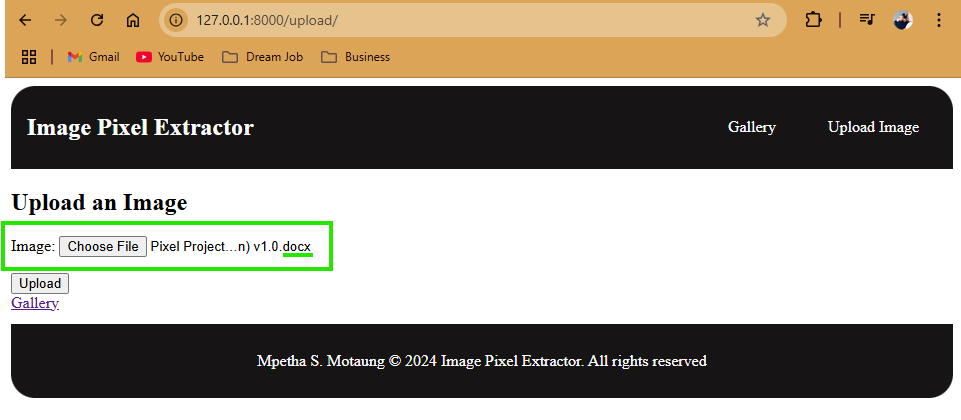
|  |  |
| --- | --- |
| Purpose: Test whether the model class is correctly handling file uploads by with defined fields. | |
| **Test** | **Expected results:** |
| Verify model field | * Check if model fields are rendered   + Served ImageField   + Served CharField |



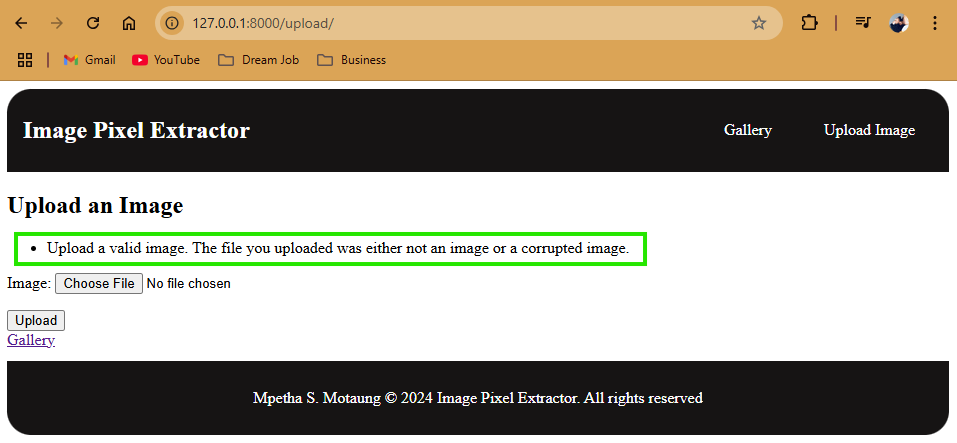
## Test Case 6

Purpose: Test whether app properly handles errors for non-image file uploads.

|  |  |
| --- | --- |
| Purpose: Test whether app properly handles errors for non-image file uploads. | |
| **Test** | **Expected results:** |
| Error handling on non-image file uploads | Throw error when user tries to upload a non-image file |



Tried to upload a non-image file (Check error handling for non-image file upload).

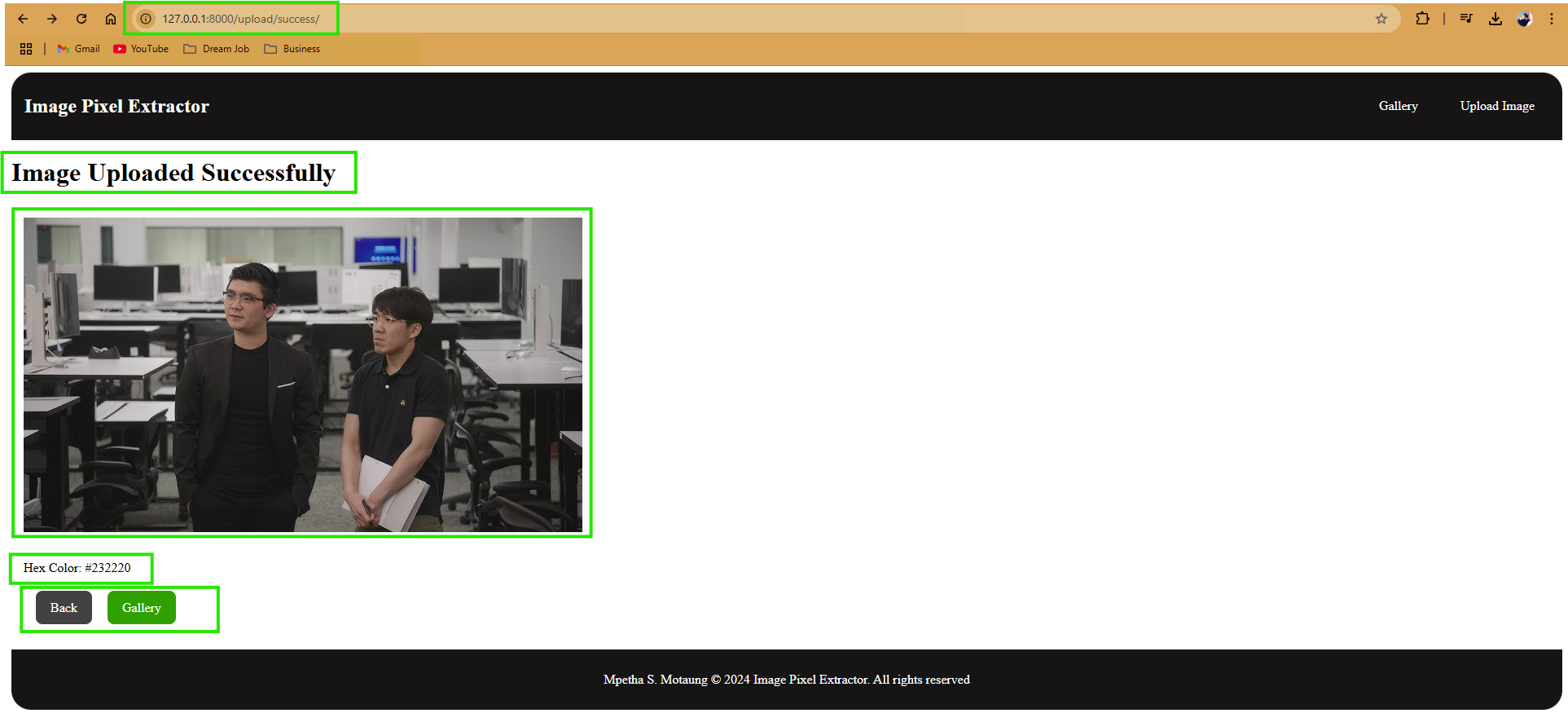


## Test Case 7

|  |  |
| --- | --- |
| Purpose: Test efficient extraction of center pixels without loading the entire image into memory. | |
| **Test** | **Expected results:** |
|  |  |

## Test Case 8

|  |  |
| --- | --- |
| Purpose: Test user flow for uploading and processing image | |
| **Test** | **Expected results:** |
| Upload success | * Upload page must display:   + Display page title   + Uploaded image   + Hex color   + Back button and gallery button (redirecting correctly) |

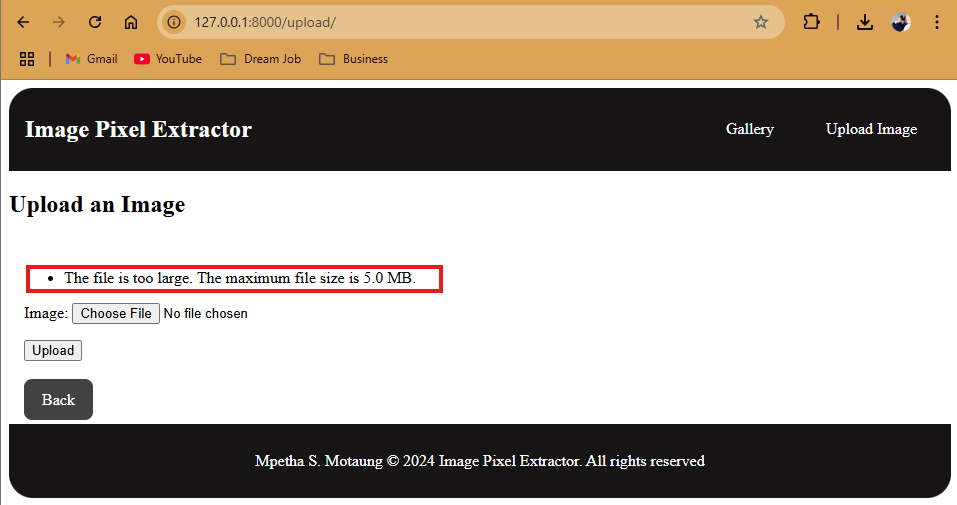


## Test Case 9

Purpose: Large image file handling

Upload image that is larger than max image size 5\*1024\*1024



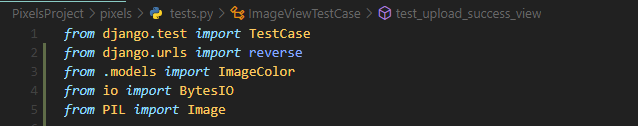


And because we added atomic transactions to the image\_upload function. The image will not automatically be uploaded to database on upload because image validation passed but file size validation failed.

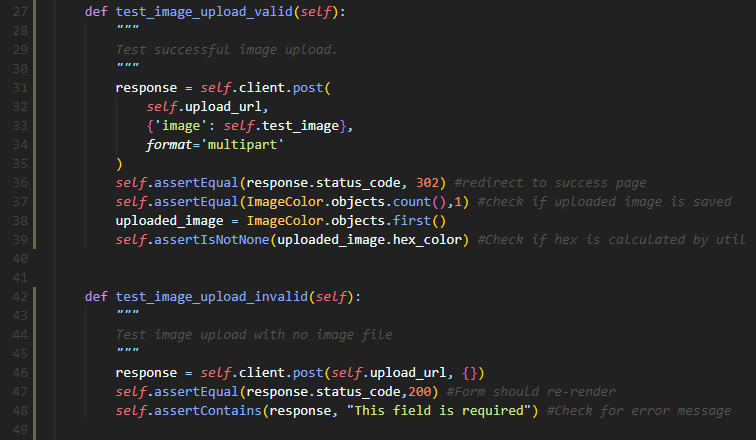
Integrations Test

This type of test is for testing different components in our application, such as between models, views, 3rd party service, etc. This test simulates user interactions with the whole system from start to finish. Testing is vital in SDLC, to ensure that applications meet the required quality standards and function as expected. It systematically evaluates software components, modules, or systems to identify errors, bugs, or unwanted behavior in development before pushing to production. []

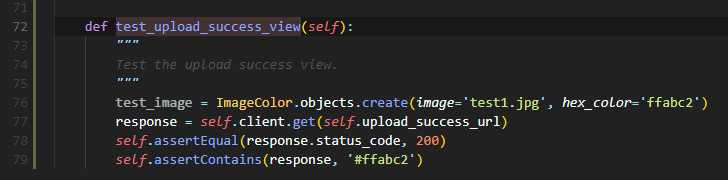
1. Write a test file for testing







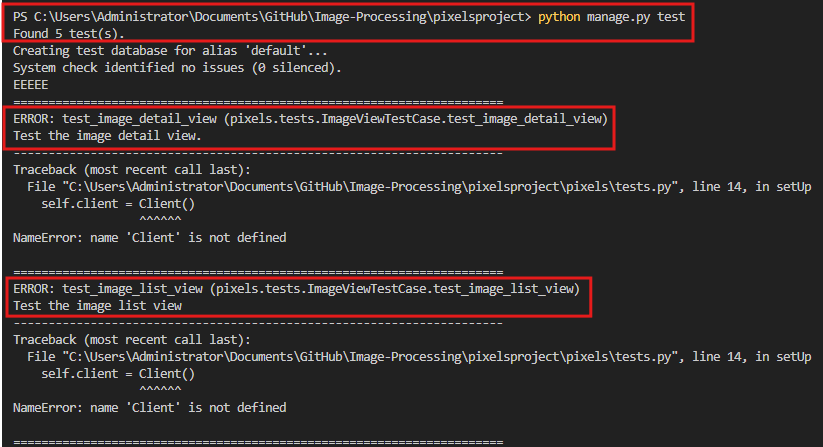


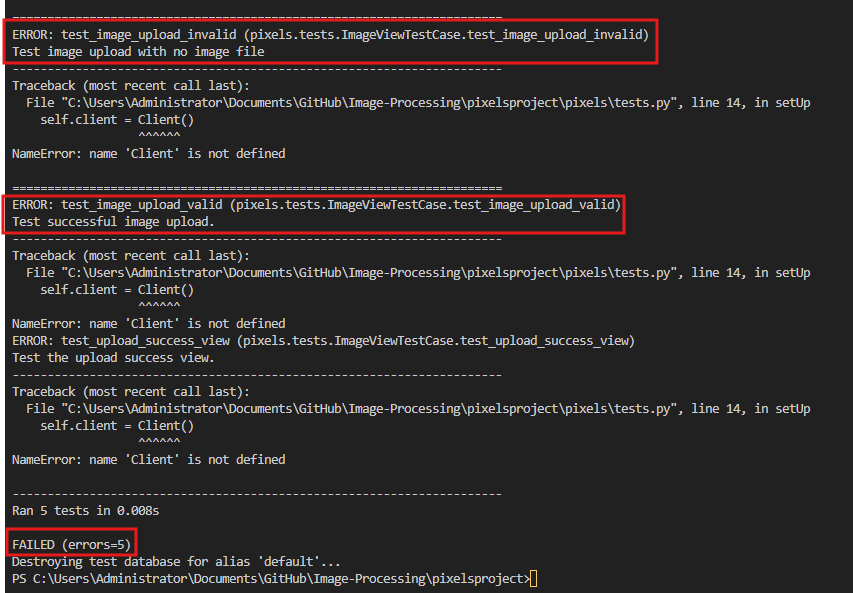


1. Run python manage.py test



Test log:





We used self.client to initiate a fresh test client and when can reuse this client to simulate multiple request across our test methods.[]

Reusability: We used self.client to ensure reusability by initializing the client in the setup, it can then be reused across all test methods in the class

Consistency: using the same client ensures a consistent testing environment within the test class

Readability: reduces redundancy because you won’t have to write client = Client() in every test method.

Testing with Selenium

Unit Tests

System tests

# CSS Styling

\*Note in dev server. When changes are made to styles.css. Clear browser cache so that updated styles are rendered\*

# Switch from SQLite3 to PostGresSQL(optional)

We can do this manually by … or automatically using PGLoader

# PIP Freeze

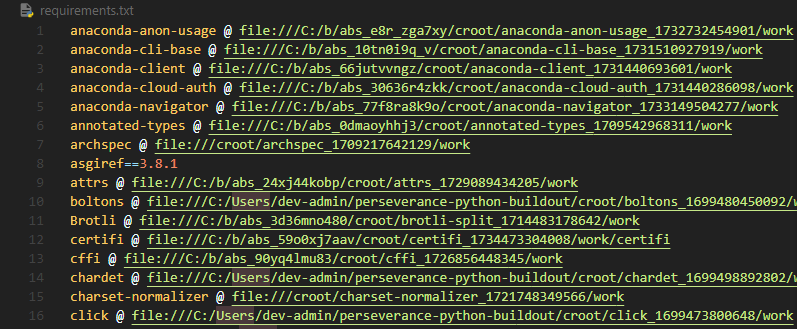
Pip freeze is a command in Python that allows you to save packages that were installed using pip in the virtual environment. It’s important because it allows you or another person to recreate the same virtual environment on another machine easily (to avoid ‘It runs on my machine’). It ensures that everyone working on the project has the same version of all packages installed. []

Navigate to project root and run python -m pip freeze > requirements



This will create a requirements.txt for all installed dependencies installed in our virtual environment. This file can later be installed on any machine to run our project.

 created environment freeze file. You can open the file to view installed packages.

Here are the requirements:  


# Conclusion

We have successfully setup our Django app and configuration. We’ve all installed all necessary dependencies (python packages). We’ve created a Model to process our image upload. We created a from to handle file uploads, image processing and extraction of pixels. We created URLs for our pixel app and registered them in our Project urls. We’ve also registered our pixels app in installed\_apps. We used the OS module to … We’ve also created templates using Django’s template system to display uploaded images along with extracted colors. We have also used anchor elements to make the image and hex color clickable and to redirect to image detail page from the image list page. We’ also implemented error handling for non-image file uploads and also implemented error handling for images larger than 5mb and also handling in cases where image processing might fail.

# Appendix A

## References

[1] Creating directories through CLI

<https://www.ibm.com/docs/en/aix/7.1?topic=directories-creating-mkdir-command>  
[] Django Debug Toolbar

https://docs.djangoproject.com/en/5.1/intro/tutorial08/#:~:text=Django%20Debug%20Toolbar%20is%20a,functions%20and%20to%20identify%20problems.

<https://django-debug-toolbar.readthedocs.io/en/latest/index.html>

[] Django shortcuts and tutorial (for handling errors)  
https://docs.djangoproject.com/en/5.1/topics/http/shortcuts/  
https://docs.djangoproject.com/en/5.1/intro/tutorial04/

[2] Installing Github Desktop

<https://docs.github.com/en/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop>

[3] Creating a repository using Github Desktop

<https://docs.github.com/en/desktop/overview/creating-your-first-repository-using-github-desktop>

[4] Create Django project

<https://docs.djangoproject.com/en/5.1/intro/tutorial01/>

[5] Django Admin and Manage.py

<https://docs.djangoproject.com/en/5.1/ref/django-admin/>

[] Django transaction – handle SQL autocommit to database

https://docs.djangoproject.com/en/5.1/topics/db/transactions/

[] Django Settings

*https://docs.djangoproject.com/en/5.1/topics/settings/*

<https://docs.djangoproject.com/en/5.1/ref/settings/>

[] Django – Testing Tools  
https://docs.djangoproject.com/en/5.1/topics/testing/tools/

[] OS Module  
<https://www.geeksforgeeks.org/os-module-python-examples/>

[ ] URL root   
https://stackoverflow.com/questions/7580220/django-urls-how-to-map-root-to-app

[] Django – Atomic transactions

https://docs.djangoproject.com/en/5.1/topics/db/transactions/

https://www.geeksforgeeks.org/transaction-atomic-with-django/

[] NoReverseMatch

<https://stackoverflow.com/questions/45724006/django-reverse-for-not-found-is-not-a-valid-view-function-or-pattern-na>

[] 403 Forbidden

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/403>  
https://stackoverflow.com/questions/36535888/django-403-forbidden

[] Django Value Error  
<https://stackoverflow.com/questions/73561751/django-valueerror-the-view-didnt-return-an-httpresponse-object-it-returned-no>

[] Django: NoReverseMatch valid view function or pattern name  
https://stackoverflow.com/questions/62834031/django-noreversematch-not-a-valid-view-function-or-pattern-name

[] TemplateSyntax Error

https://stackoverflow.com/questions/19428572/django-templatesyntaxerror-could-not-parse-the-remainder

[] How to deliver images (Django Forum)

<https://forum.djangoproject.com/t/whats-the-best-way-to-deliver-images/27525>

[] Django templates

https://docs.djangoproject.com/en/5.1/ref/templates/language/

[] Django root urls (Django Forum)

https://forum.djangoproject.com/t/order-of-includes-in-root-urls-py/23921

[] Writing and Running Tests in Django

<https://docs.djangoproject.com/en/5.1/topics/testing/overview/>

[] Colors on an Image (Pillow)

<https://www.geeksforgeeks.org/python-pillow-colors-on-an-image/>

[] Why declare app\_name in app URLs

https://stackoverflow.com/questions/61254816/what-is-the-purpose-of-app-name-in-urls-py-in-django

[ ] How to create a conda virtual environment (CLI)

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

[ ] Create Static folder and Media Folder (static is responsible for housing css, js, html and images). Media folder is responsible for housing media uploaded by the user

https://docs.djangoproject.com/en/5.1/howto/static-files/

[ ] Image Field

<https://docs.djangoproject.com/en/5.1/ref/forms/fields/>

[ ] Image Upload Form

https://docs.djangoproject.com/en/5.1/ref/forms/api/

[ ] Bind Uploaded files to form

<https://docs.djangoproject.com/en/5.1/ref/forms/api/#binding-uploaded-files>

[ ] Pillow (needed for Image Field usage)

<https://pypi.org/project/pillow/>

[ ] Pip Freeze

https://pip.pypa.io/en/stable/cli/pip\_freeze/