

Zainab

Zainab Abbas

19/12/2024

202001012





**Contents**

Cover Page ...................................................................................................................................................................................... 1

[Problem Statement Definition – Task 1 3](#_Toc185541905)

[Report 3](#_Toc185541906)

[Reflection 21](#_Toc185541907)

# Problem Statement Definition – Task 1

**Motivation:**

The beauty and makeup industry is vast and constantly expanding, with many people being fans and users of the products and services offered by this industry to have a need to be offered reliable and engaging platforms that would allow them to share information, tips, and feedback on products and services. However, a specific standalone solution is the most required to offer control and configuration of the activated blog and the ability to perform integrated communication with the other users.

This project is motivated by the need to design an easy to use and interactive software which will allow all the users to upload their content related to makeup, manage posts effectively and get attention.

**Project Background / Objective:**

The purpose of this project is to design a web application in the makeup blog genre, which is based on Python, Django and MongoDB, for creating, editing, and publishing makeup articles. It should allow the users to log in somehow securely create, edit their blogs and easily locate their posts. The big idea here is to ensure that users have great experiences fully backed by well-developed platforms.

**Functionalities:**

**Add/Delete/Edit Blog Posts:**

Users can start with new blog posts as well as edit their existing blogs and even remove posts that are not necessary. This functionality guarantees the users complete discretion on the content shared on this platform.

**Login/Logout System:**

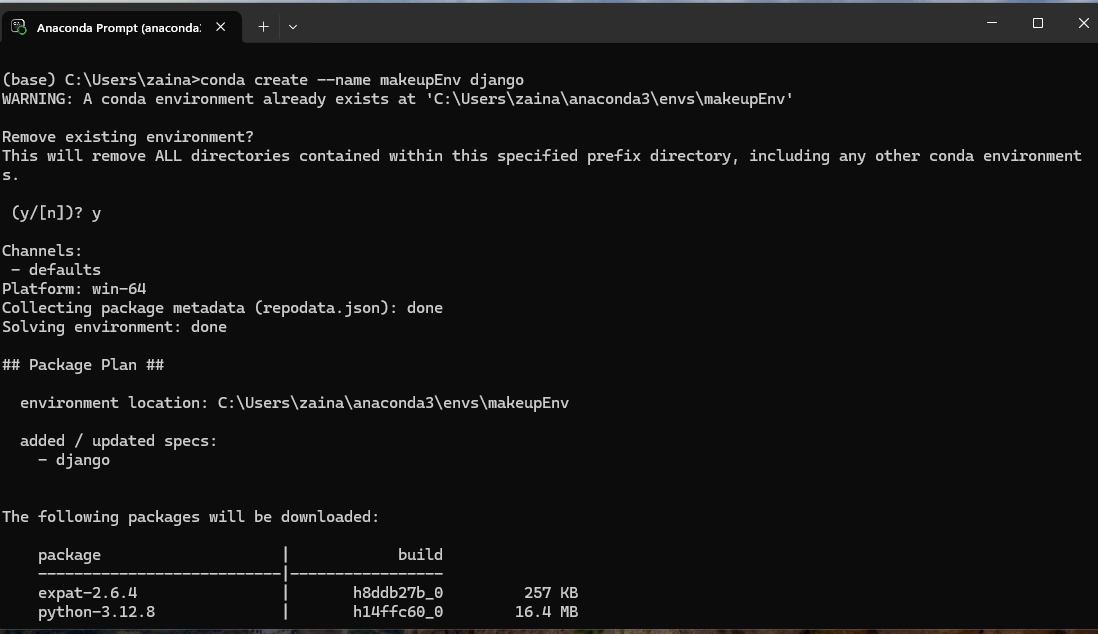
User authentication is integrated to the extent that users can log in and log out of the application for safety purposes. This helps protect data and also provide individuals with the unique ability to use the site.

**Search Feature:**

It also contain a search box where you can enter the name of any blog or the topic you are interested in. It increases the company’s navigation power and its ability to deliver better user experiences in the sense that it provides fast-linking to contents.

# Report

Step1: Creating a project environment using the following command:



Step 2: Activating the created environment following this command:

A screenshot of a computer

Description automatically generated

Step 3: install the given applications one by one:

certifi==2022.9.24

cffi==1.15.1

charset-normalizer==2.1.1

cryptography==38.0.3

defusedxml==0.7.1

Django==2.2.8

djongo==1.3.6

dnspython==2.2.1

idna==3.4

oauthlib==3.2.2

Pillow==9.3.0

pycparser==2.21

PyJWT==2.6.0

pymongo==3.12.1

python3-openid==3.2.0

pytz==2022.6

requests==2.28.1

requests-oauthlib==1.3.1

sqlparse==0.2.4

urllib3==1.26.12

Step 4: Step 4. Cd (change directory) to where you extracted the folder:



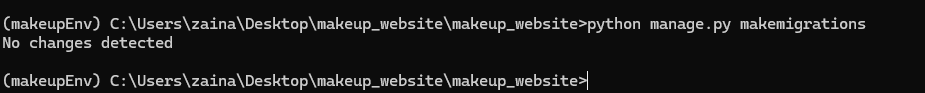
Step 5: Create a superuser:

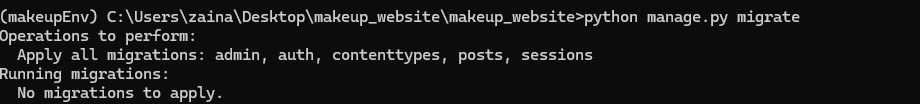
Using this command: python manage.py createsuperuser

A computer screen with white text

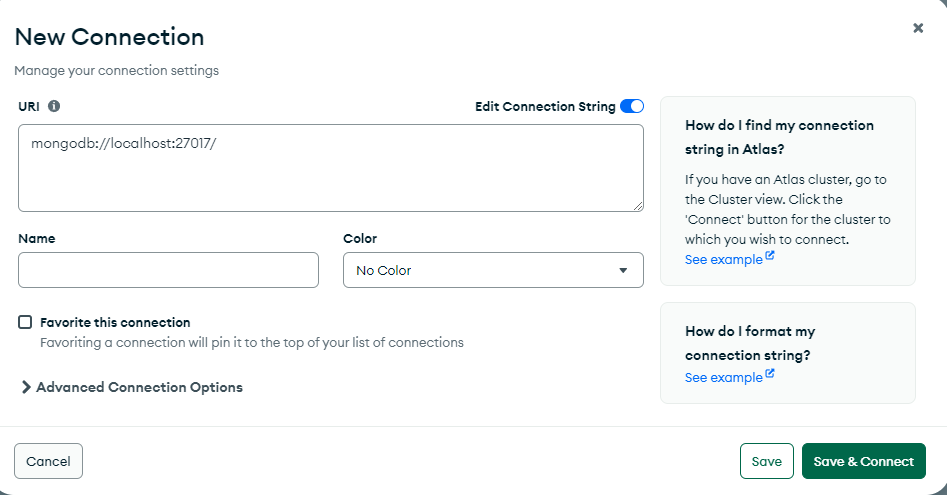
Description automatically generated

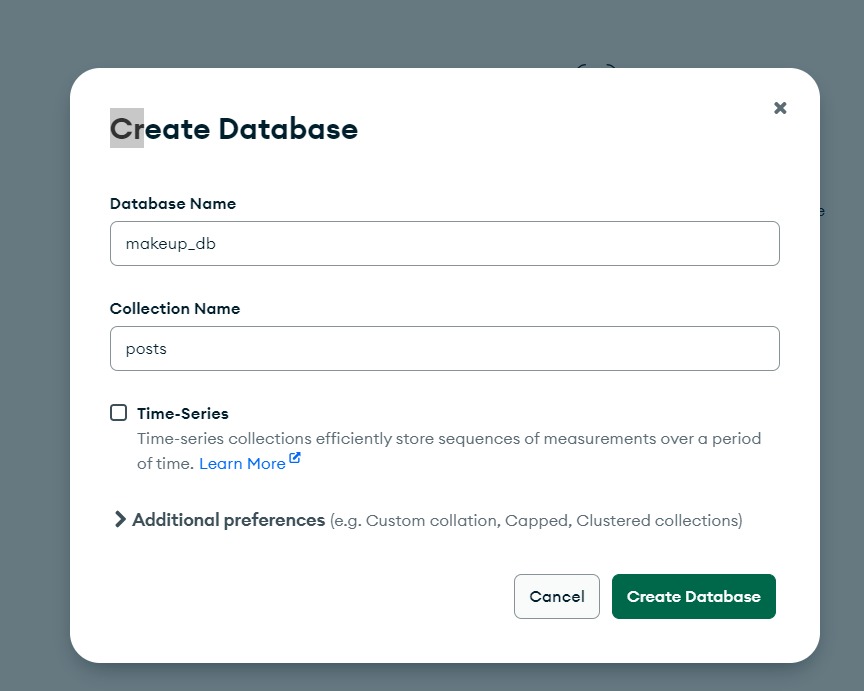
Step 6: make migrations then migrate:



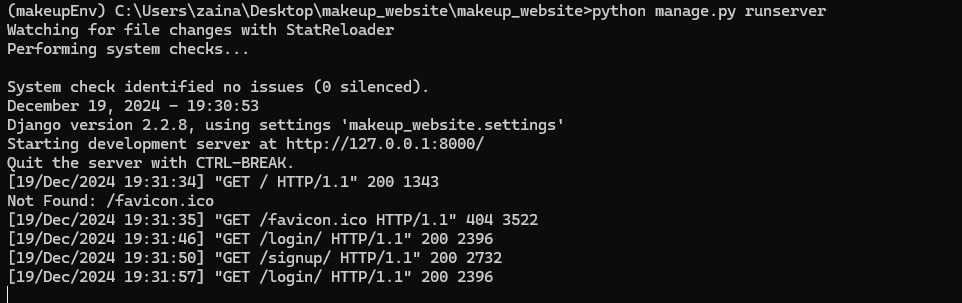


Step 7: open your compass and do as following:





Step 8: run the server:

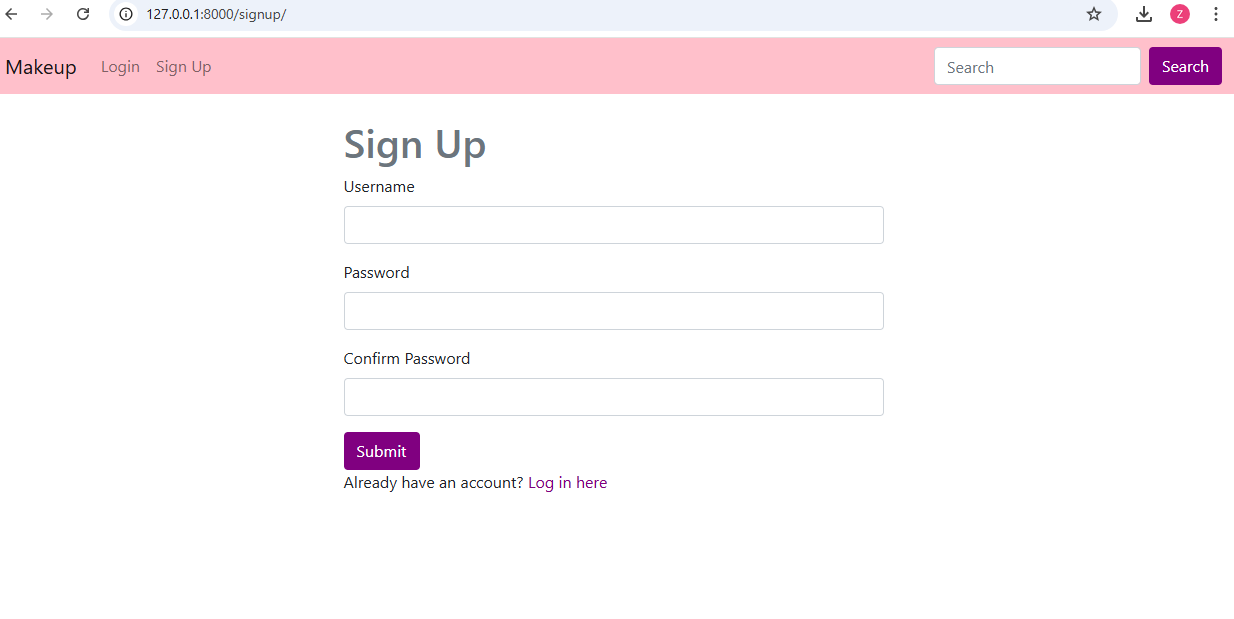


Copy the url http://127.0.0.1:8000/ and paste it in any browser.

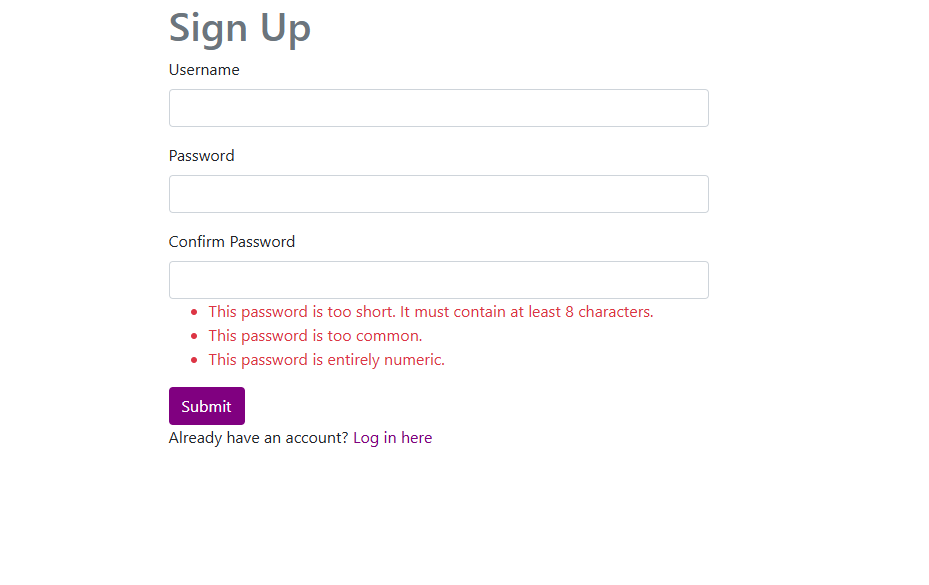
When a normal user opens the website this is what they see:A screenshot of a computer

Description automatically generated

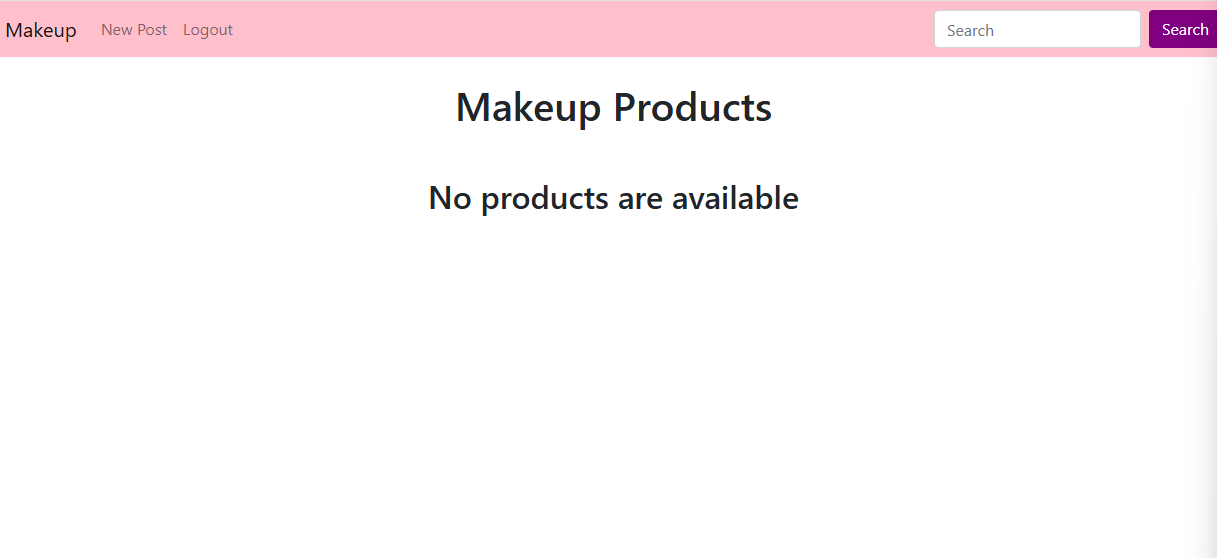
You won’t be able to add a car unless you’re registered and logged in:



The password should have 8 characters and should be mixed.



After registering this page will appear:



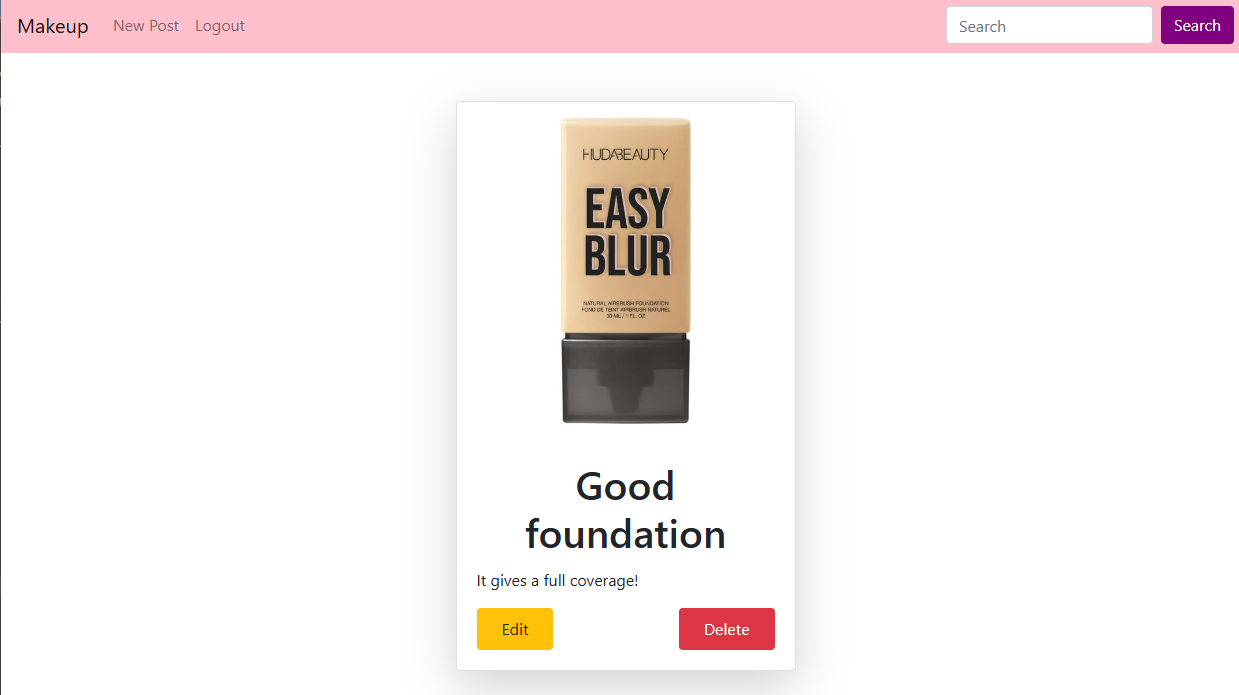
As you can see the “New Post” appears.

You fill in the the title, description, and add a picture based on the desired information, when done and added everything should appear in the home page.

A screenshot of a computer

Description automatically generated

Once you fill in everything this is how it should look.



**Codes:**

**Makeup\_website(main):**settings.py:  
 The Django’s settings file contains the `INSTALLED\_APPS` section that points to the applications active and used in your project. It consists of Django standard applications such as admin application, authentication application, session application, messaging, and static application. Further, it maps your app, in this instance ’posts‘, or the logic and architecture of your project which comprises models, Views, templates among others. By listing these apps, Django combines their functionality, so that these apps can interact within the project.

A screen shot of a computer screen

Description automatically generated

The BASE\_DIR variable determines the base directory of your Django project by using the settings.py file path only to go up two levels. This is used as a reference point when relating to management of project files and directories.



The DATABASES dictionary used for setting up and configuring the databases connection. It specifies djongo as the database engine to connect Django with MongoDB and provides the necessary details, including the database name (makeup\_db) and connection settings like the MongoDB URI (host), retry options, and write acknowledgment (w: 'majority'). In total, all these configurations help the project to communicate with the MongoDB database.

A computer screen with text

Description automatically generated with medium confidence

This code fix app configuration for static files and authentication redirection. STATIC\_URL defines the base URL for the server to open static file and STATICFILES\_DIRS defines the directory for place of custom static files. LOGIN\_URL determines the URL of a login page, LOGIN\_REDIRECT\_URL shows that to which URL the users will be redirected after login and LOGOUT\_REDIRECT\_URL shows to which URL the users will be redirected after logout.

A screenshot of a computer program

Description automatically generated

Urls.py:

This code tell django where to show the different parts of this project on the indetant web address. Most of the paths are very straightforward: it includes path for the admin interface, admin/, and any other paths will redirect them to the posts app. It also serves media files provided the project is in debug mode which appends a static URL pattern for media during the moment of development.

A screen shot of a computer program

Description automatically generated

**POSTS(APP):**

admins.py:

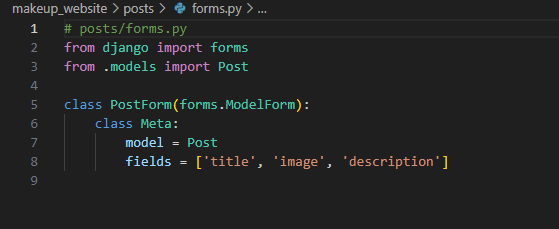
This code determines the format, in which posts are displayed in the Django admin panel so that the admins can work with them without difficulties. Here it displays the post title, user ID and created on date of the particular post on even the listing page of the posts. It also allows admins to learn about the posts they have written or any other new posts by searching the title and description of posts and also by selecting newly posted, most posted, or any particular desired author. These customizations assist the admins to easily navigate through the posts unlike having to navigate through every post. The great thing is that this code has the last line which make sure that the `Post` model will be using these custom settings in the admin area.

A screen shot of a computer program

Description automatically generated

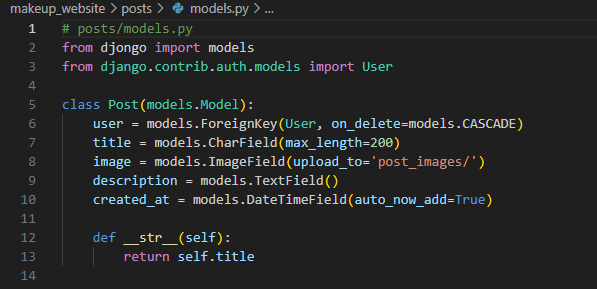
forms.py:

This file contains a form for posting and editing posts. Post model tells which fields of the form is acceptable, such as the title, an image, and the description. The `PostForm` class functions like a template to guarantee that when someone posts information on the website, the information is well checked and stored. It is used in the views as they take charge of rendering the form on the webpage and also taking the input from the user.



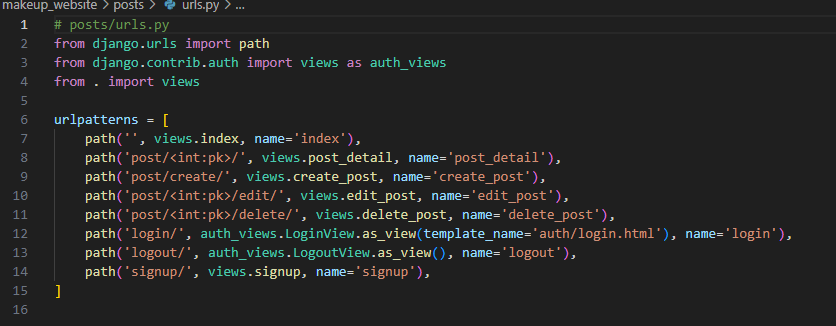
models.py:

These lines of code give the structure of a “Post” in the application. It defines the fields that a post will consist of for example the title, the image, the description and the date of posting. The post it is related to a certain user by a `ForeignKey,’ which means that one post belongs to one user. The concept of this file is to inform Django where and how to store posts and provides a simple and effective way to obtain the title of the concrete post anytime (with the help of the \_\_str\_\_ method).



urls.py:

In the FOSS context, it must be established which type of web page is be created, and the kind of action that is expected to occur when the page is accessed. For instance, if a user is on the home page (), they can only see the list of posts. If they visit the pathname they will be able to post a new post by visiting /post/create/. It also defines URLs for viewing, editing, or deleting post which is meaningful to the application. Also, this file deals with routes of user login, logout and signup so that a user can be in a position to handle an account. It is in fact the blueprint of traffic management within the website.



views.py:

This file deals with deciding what the user actually sees and with which he/she can interact on the site. For instance if the user is in the index view – they are presented with all posts or those posts that have been searched. The signup view also enables the new users to sign up and they’re immediately signed in. The post detail removes and shows a single post from the blog when a user click on any post. The different views that are create\_post, edit\_post and delete\_post enables user to create a post and edit a post and also delete a post if they are logged in. These views control the data that passes between the user and the forms in addition to the templates used to present the content on the webpage.

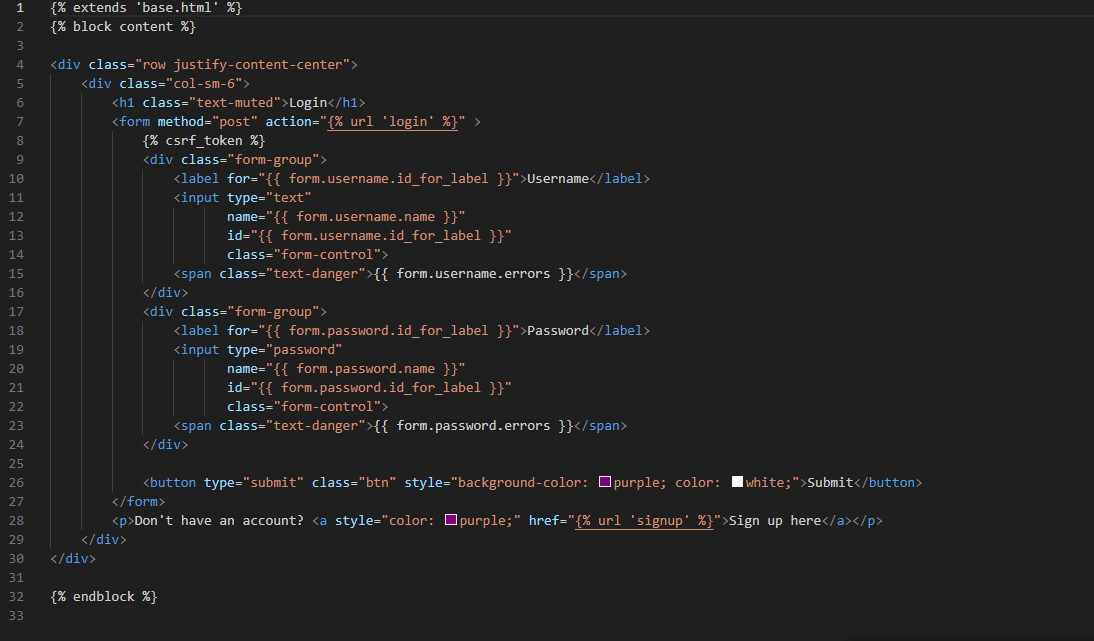


**Templates:**The functionalities are going to be explained by these html codes:

Authentication(auth):

Login:

This file is actually the HTML file that will be used to display the login page. It extends a ‘base.html’ template, which most probably has the common layout of the site, like the top and the bottom lines. This page is created inside the `{% block content %}` where the page has a centered ‘log in’ form. It contains two input fields: The user’s first account should be the username, and the second account should be the password. They both hold the corresponding form fields via the Django’s form system (i.e., form.username and form.password). Each field has an error message that occurs in red within the textual content when the consumer input an invalid input. The form specify the method of sending as “POST” when the user submits the form to the server. Also used is a `{% csrf\_token %}` tag that will prevent what is known as cross site request forgery, a kind of security hack that will see the request being made by an impersonator and not the actual user. At the bottom, there is a signup button which takes those users who have not signed up to the site a direct access to sign up.



Logout:

The content of this page pops up when a user navigates out of the running application. As does the other pages, it is build from the `base.html’ template. The content is represented just by the message informing the user that he has been logged out, and offering the link to allow the user to log in again. This is a minimalist page utilized only to remind the user of their current status, being logged out of the application, and offer them a method to re-login.

A computer screen shot of a computer code

Description automatically generated

Signup:

The signup.html is to render the user registration page. Similar to the login page, it inherits from the base.html and orients the signup form at its middle. This form requires the user to enter a username, password and then repeat the password for confirmation. The error messages for validation occur beside each field label for the form that will appear in red in case the input provided by the user does not meet the specifications for the requirements such as password and username. The form uses the POST method to send the data, here we have incorporated {% csrf\_token %} for security against illegitimate users and malicious sources. Once the user signs up by filling the form, they are registered in the system. At the bottom of the form there is a link for those users who already have an account to sign in, which leads to the sign in page. This is helpful for when the user is still on the page, but has somehow missed the login step because it leads them straight to the login page.

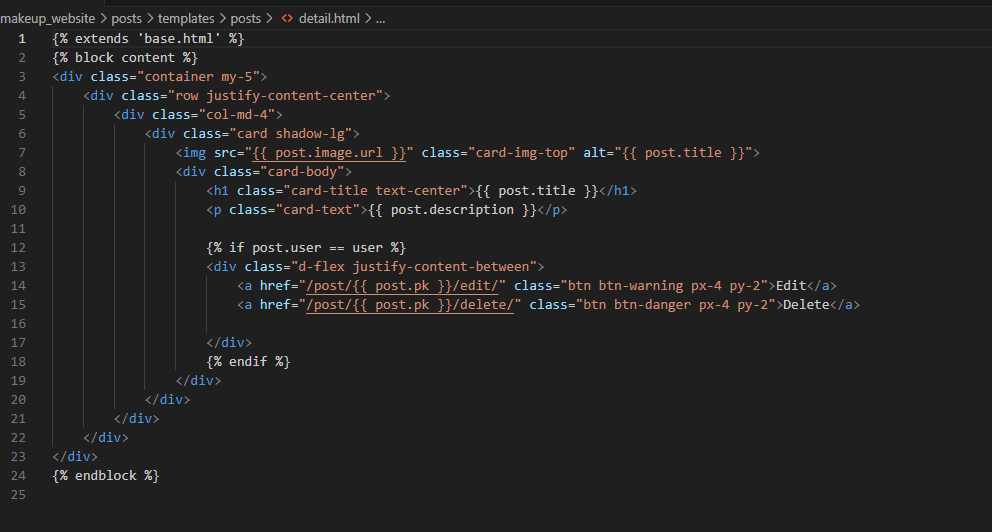
A computer screen shot of a program code

Description automatically generated

Posts:

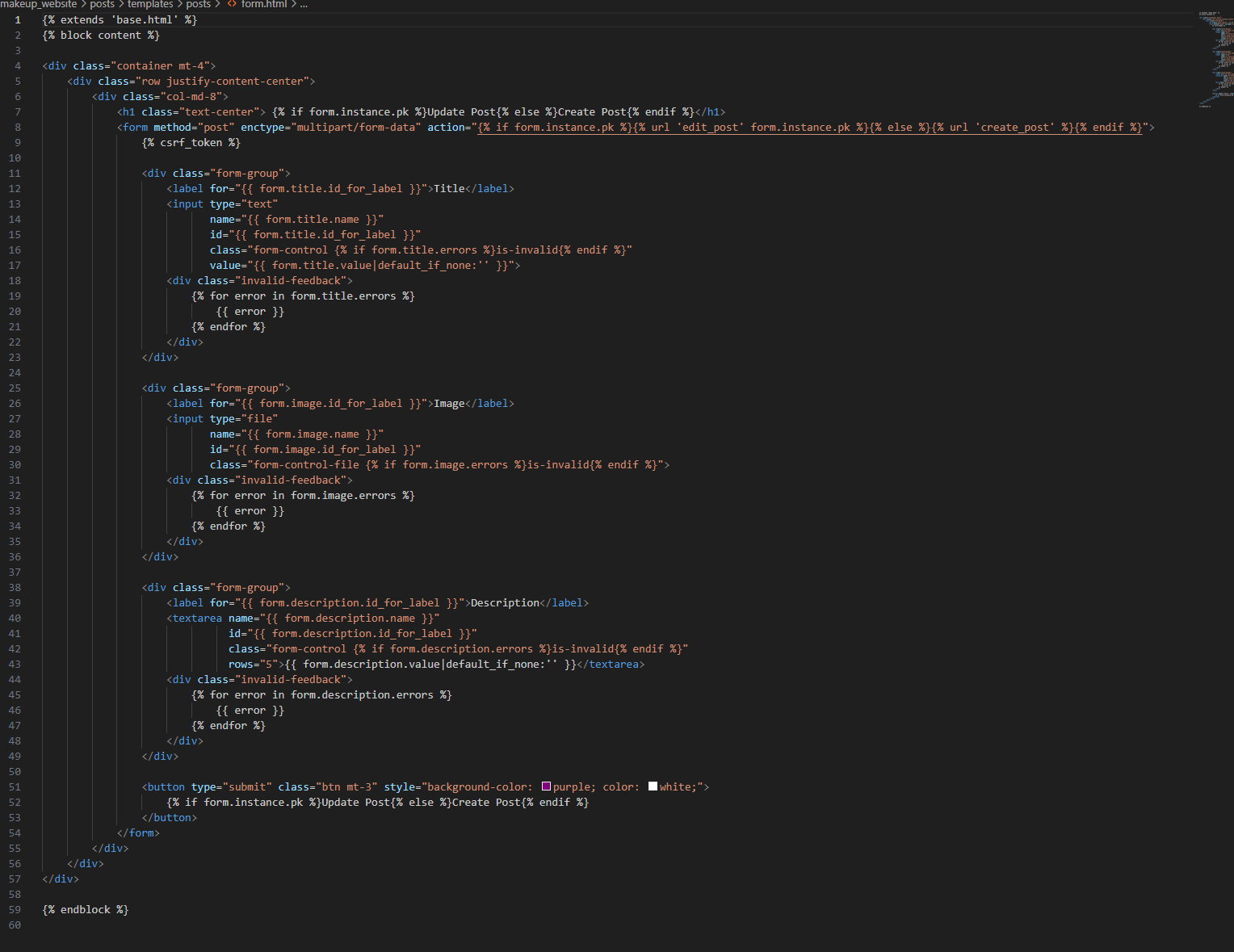
Detail:

This page shows the details of a single post in a clean and organized way. It includes the post's image, title, and description, all displayed inside a card. The person who created the post can also see buttons to edit or delete it, making it easy to manage the content directly from this page. The layout and design are built using Bootstrap to ensure it looks good on different devices.



Form:

This page is a flexible form for creating and updating posts. It figures out if you’re working on a new post or editing an existing one and adjusts the page title and behavior accordingly. The form is simple and user-friendly, it has fields for the title, image, and description. If you make a mistake, it displays the issues and shows error messages where the problem is. The Submit button guides users to save a new post or update an existing one.



Index:

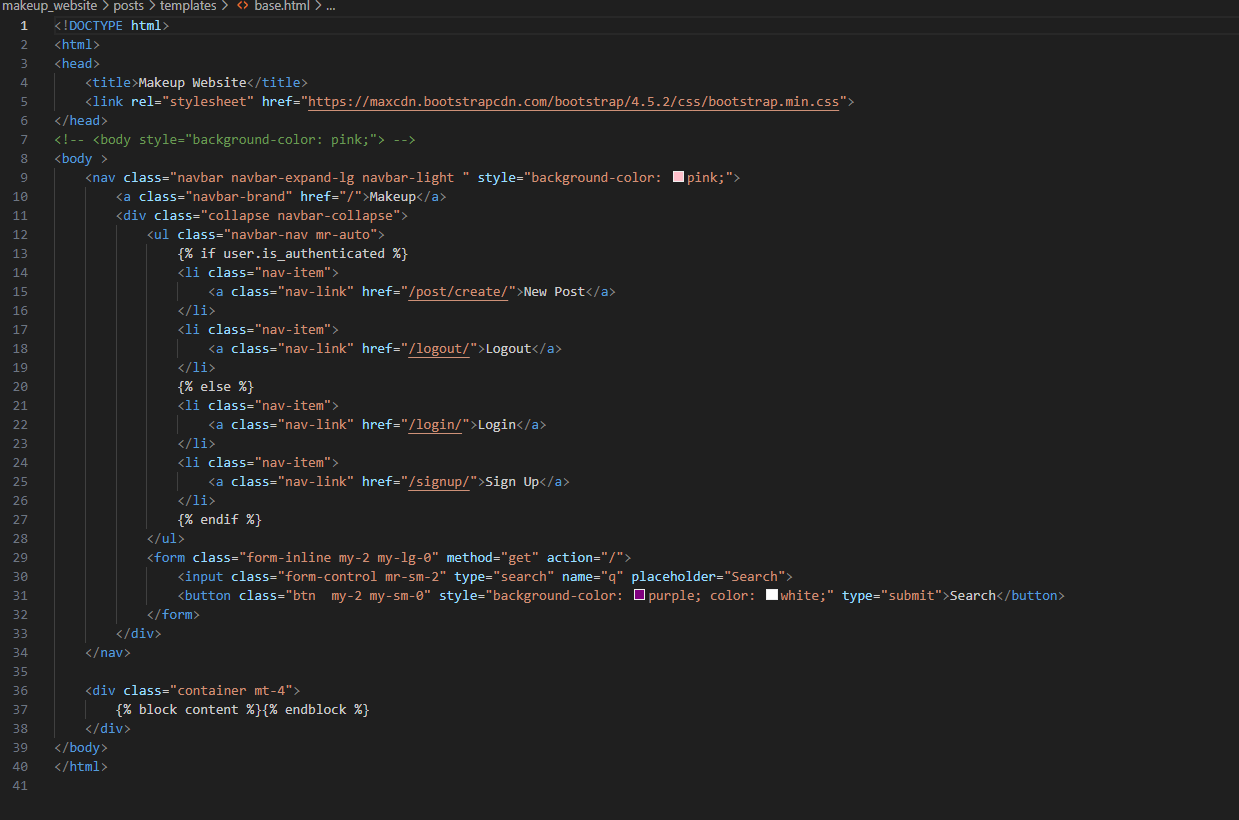
This page acts as a gallery of makeup products, it displays each item with an image, title, and buttons for actions. It’s organized in a grid format and it’s visually appealing and easy to browse. Users can click "View Details" to view the details of a product. The person who created the post can also see buttons to edit or delete it, which gives them control of the content. If there are no products to show, a message will be displayed. It’s a simple and functional way to show products.

A screen shot of a computer

Description automatically generated

Search:

This template is like the backbone of the website. It gives every page a consistent look with a pink header, easy navigation links, and a search bar. Depending on whether you’re logged in or not, you’ll see different options in the page. The rest of the page’s content changes depending on where you are, but it all fits nicely into this friendly, stylish design.



# Reflection

Looking back on the journey behind creating the Makeup Blog, it’s been exciting and fulfilling. To put together the makeup enthusiast's digital playground of a platform, the project involved three apps—Post, UserAuth, and Search. The site was both enjoyable and functional to use because each app played a critical role on the site.

Post App:

The platform is built around the Post app, which lets users create and share posts about creation makeup products. The idea behind it is that it would take a snapshot of all the most important info there, such as titles, descriptions and images, while presenting everything in an attractive, easy to view manner. The site has been styled using Bootstrap to get a new, modern feel.

The best feature of this app is that it allows users to handle their posts. It’s easy to add new content, edit existing posts, or delete stuff that’s been around for too long: the site stays fresh with up to date, interesting content.

UserAuth App:

With the UserAuth app People feel personal and secured in Makeup Blog. This is great so that users can sign up for an account, log in, and interact with the platform in a much more meaningful way. After logging in users have the opportunity to create posts just as they wish and have access to personalized features with the assurance of the security of their information. Smooth redirection after login or logout is an addition to your thinking, which greatly improves the overall experience and adds to an easy way to navigate around.

Search App:

The whole point of the Search app is about convenience. Instead of it, it provides users an easy way to find out the product they are searching for and by typing a keyword like a specific product or a category of makeup that the user wants to see. This makes for consistent and intuitive experience from rest of the site as well as from the search results.

Final Thoughts:

The Makeup Blog has come about as the contact between these apps has become a vibrant and interactive space for beauty enthusiasts. Of course, in the process we had to fix some bugs, sharpen some features and make sure everything was working in the right manner. And those challenges showed us something, they helped us learn something, they helped us grow our skills.

That's been a good example of how powerful teamwork, and some pretty good tools, like the Django framework, can be. Most of the time, there’s always room for improvement — perhaps by adding product reviews, ratings, or community discussions and all that — but still, the blog is a great one considering it’s a great way to add some spark to the makeup world and unite makeup lovers.

And I’m proud of what we’ve built and excited for it to continue to be helpful to the beauty community in the future.