

# **DEGGENDORF INSTITUTE OF TECHNOLOGY**

**Faculty of Electrical Engineering, Media Technology and Computer Science**

## **Programming World**

**Programming Assignment – Advanced Programming Techniques**

**Summer Semester 2018**

### **Collaborators in this project:**

- |                  |          |
|------------------|----------|
| 1. Johnson James | - 730801 |
| 2. Juhi James    | - 730816 |
| 3. Nikhil Baby   | - 775766 |

## Table of Contents

<b>1. Introduction .....</b>	<b>3</b>
<b>2. Working of Programming World (installation and usage) .....</b>	<b>3</b>
2.1 Running the server.....	3
2.2 Creating a new user account.....	5
2.3 Log in using user credentials.....	7
2.4 Write a blog.....	8
2.5 Browse the blog pages .....	9
2.6 Type of users.....	11
2.7 Commenting and deleting blog post... ..	11
2.8 Admin log in.....	12
2.9 Activity flow diagram .....	14
2.10 Use case diagram .....	15
2.11 UML class diagram .....	16
<b>3. Code Documentation .....</b>	<b>19</b>
3.1 Settings.....	19
3.2 Models... ..	20
3.3 Forms... ..	21
3.4 Views... ..	22
3.5 Templates... ..	25
3.6 Urls... ..	28
<b>4. Deviation from Plan.....</b>	<b>29</b>
<b>5. Reference .....</b>	<b>30</b>

## 1. INTRODUCTION

The project is a blog-spot for programming enthusiasts. We focus on enhanced user experience combined with a rather minimalistic approach to ease the burden of sophistication for amateur bloggers. Subscribing to this blog will be free of charge so that users can enjoy writing and reading blog posts. We have included three web apps for programming languages: C/C++, Python and Django. The bloggers can publish their content under these three categories. The user should need a profile in the website to publish their posts through which the blogger can write a blog of his/her interest.

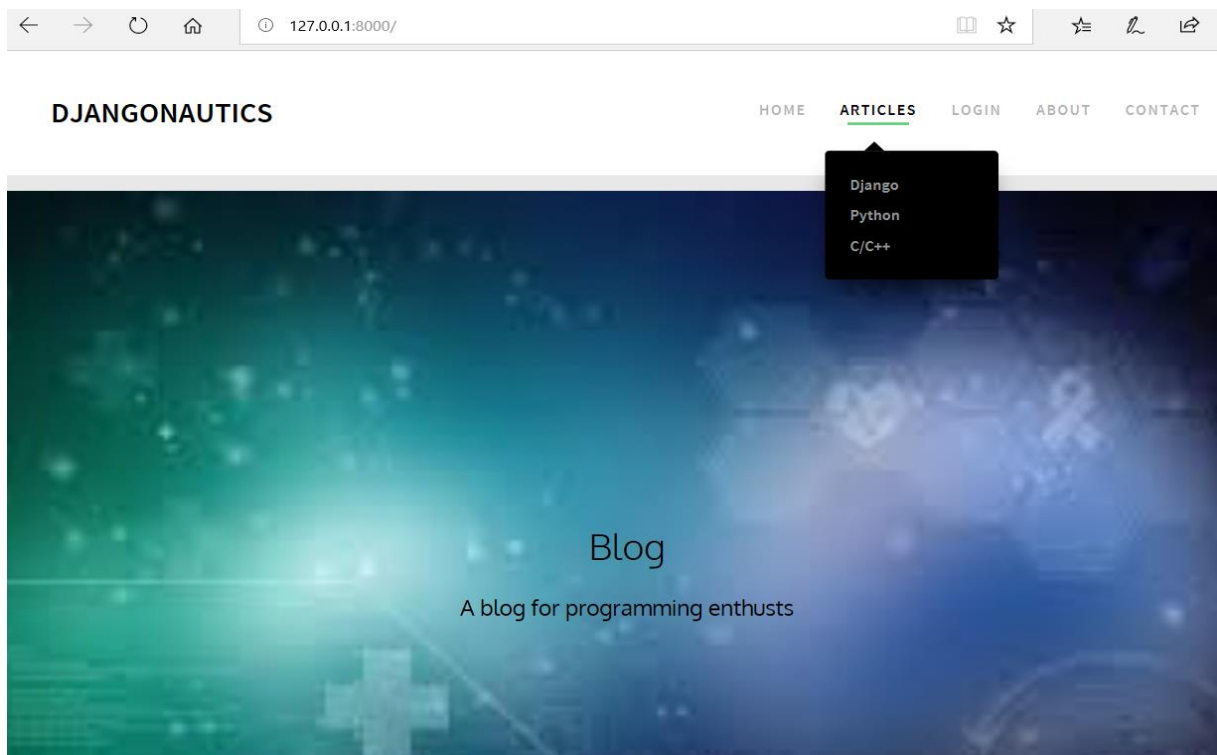


Fig 1. All articles will be displayed in the homepage regardless of its category.

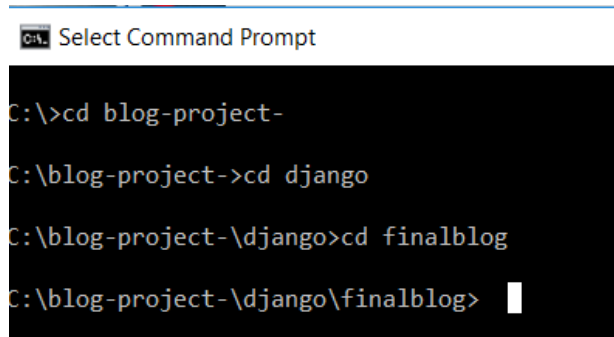
## 2. Working of Programming World

### 2.1 Running the server

At first the project files should be pulled from the git repository to run the website. The link of repository for Programming World project is

[git@mygit.th-deg.de:jj23816/blog-project-.git](https://mygit.th-deg.de:jj23816/blog-project-.git)

Once the files are pulled to the local system the website should be made to a functioning site for this we need to run the website locally in our personal computer or laptop. for this we need to access the local host and the website up and running. We need the local computer should be pre-installed with python and Django. The commands can be access through command prompt or any command windows of choice. During the sample run we used command prompt as shown in the Fig.1.

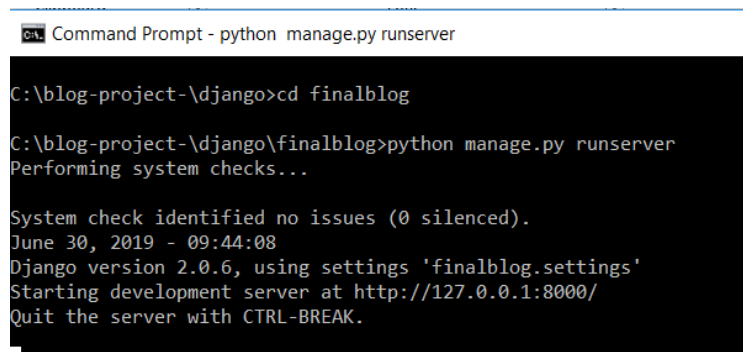


```
C:\>cd blog-project-
C:\blog-project->cd django
C:\blog-project-\django>cd finalblog
C:\blog-project-\django\finalblog> |
```

Fig.2: Command to runserver

First, we need to access to the source directory where which the files are pulled. The file here is saved in the C drive under the name blog-project-. (C:\blog-project-\django\finalblog)

Once the file path is obtained we give the command to run the server which is “python manage.py runserver”. If the command is executed successfully a development server is created with IP address <http://127.0.0.1:8000/> as shown in Fig. 2. This confirm that the website can be accessed through a web browser of choice through this IP address. the development server can be quitted by using CNTRL+C. The IP address is given in the browser which will direct to the Programming World home page as show in Fig 3. The home page displays all the blog posts created by each author under different topics where the readers can read the content of the blog without requiring a profile in the blog spot.



```
Command Prompt - python manage.py runserver
C:\blog-project-\django>cd finalblog
C:\blog-project-\django\finalblog>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
June 30, 2019 - 09:44:08
Django version 2.0.6, using settings 'finalblog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
|
```

Fig. 2: Development server creation

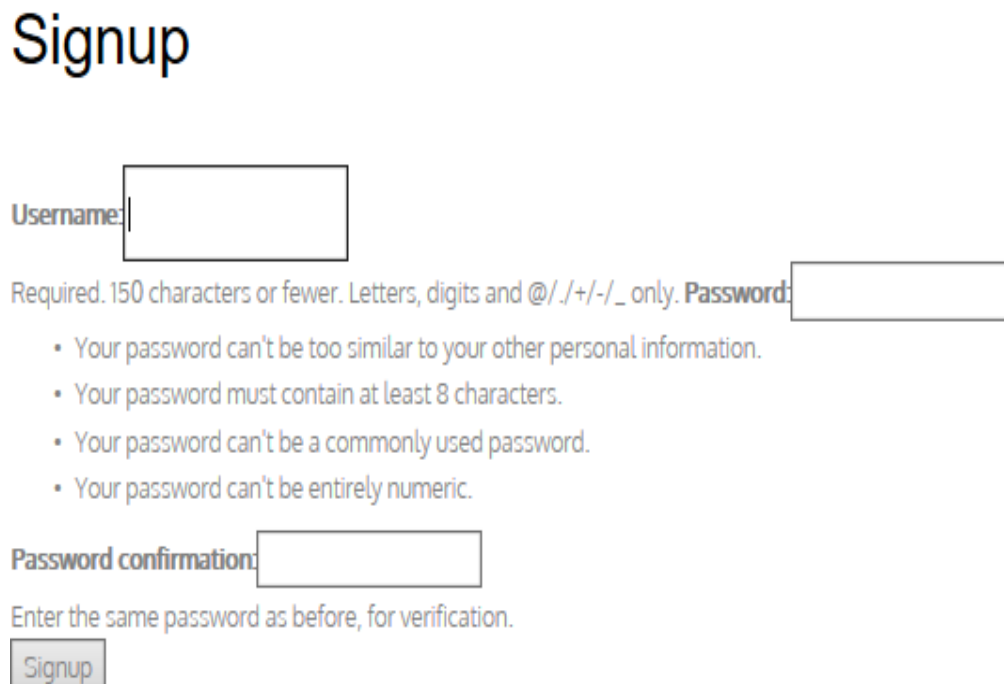
## 2.2 Creating new user account

The contents of the blogs are also arranged in the descending order of the date of publishing. In the home page the navigation bar has following tabs home, login , articles, about and contact us.



Fig. 3: Home page

To create a profile, use the sign-up tab which will redirect to the sign-up page <http://127.0.0.1:800/accounts/signup/>, Fig 4. In the sign-up page the user should provide the following information to get started a new account in the Programming World, which are a user name of choice, his/her first name, last name and E-mail address.



The image shows a web form titled "Signup". It contains a "Username:" label followed by a text input field. Below this is a "Password:" label followed by a text input field. Under the password field, there are four bullet points listing password requirements: "Your password can't be too similar to your other personal information.", "Your password must contain at least 8 characters.", "Your password can't be a commonly used password.", and "Your password can't be entirely numeric." Below the password field is a "Password confirmation:" label followed by another text input field. Under the confirmation field, it says "Enter the same password as before, for verification." At the bottom left of the form is a "Signup" button.

Fig. 4: Sign-up page

The user should also provide a password meeting this criteria's

- 1.the password cannot be similar to your other personal information's
2. should contain 8 characters
3. Should not be commonly used password and
4. cannot be entirely numeric.

If the password does not meet the following criteria's there will be an error message to re-enter a new password. Once the password is chosen it should be reconfirmed by retyping the password in the password conformation box. The sign-up process is then complete after confirming sign-up button. To quit sign-up process the user can use cancel button to get redirected to the home page.

## 2.3 Log in using User credentials

Filling in the sign-up form and confirmation will create the user a profile in the Programming World website and redirect to the home page. The sign-in tab in the home page will direct to the login page, Fig 5. Where the user can enter the username and password to access his/her profile. In case the when user forgot the password, there is a Forgot Password option in the sign-in page. Clicking this option will redirect to a new window where the user is asked to enter their registered E-mail in the dialogue box. Once the E-mail is confirmed to be registered to the profile, a reset link will be sent to the running command window by which the user can reset the password.

DJANGONAUTICS

HOMEARTICLESLOGINABOUTCONTACT

Blog

A blog for programming enthusiasts

Log in

Username:

Password:

LOGIN

Not got an account? [Sign Up](#)

Fig. 5: Login page

## 2.4 Write a blog

After login the page is redirected to homepage where the user can author a new bog or can check on other blogs from this page. The user also can check his profile details from the user information button un the top left corner of the page Fig.6. The user can also logout from his profile using log-out tab after his session. The blog editor can be accessed using “New Article” button.

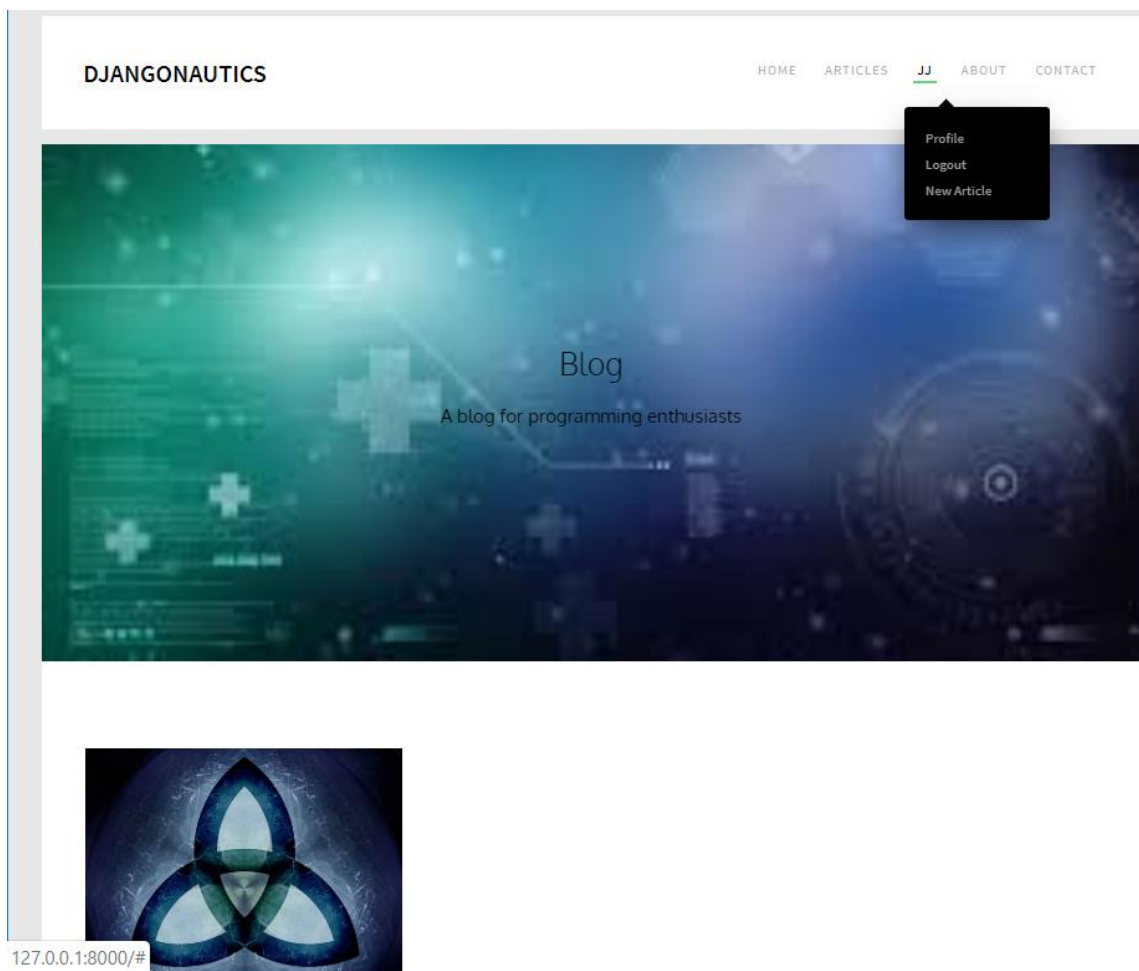


Fig.6: Profile page

The “New Article” will open a new editor window, the blog editor consist of the following boxes the title of the post, author, subject and content box.



## New Article

Author:

Title:

Body:

Thumb:

Types:

Fig. 7: Blog write page

The subject will determine whether the blog is classified under which category of the blog and also get posted under the specific blog category page. This make the readers easy to access the blog based on the nature of the content. The blogger can also upload picture into his content using the upload option provided in this page the user can select the images from the local drive using the browse option during upload. Once the content is ready to post the user van post the blog by clicking on save button and If the blog is posting successfully a success message will be displayed on the page.


## 2.5 Browse the blog pages


The user or other website visitors can read the blog without any profile under each category. The pages of each categories are shown in Fig. 8. These categories can be chosen from every page. The


**CONTACT US**

*Far far away, behind the word mountains, far from the Home*

**Our Address**

 Kolpingstraße 1

 +4965 2355 98

 info@thd.com


Name

Email

Message

blogs are sorted based on the date of posting. the reader can see the post title, date, subject and author name in this window.

**UPDATE YOUR PROFILE**



**JJ**

JJ@gmail.com

Username:  Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Email:

Image: Currently: [default.png](#)

Change:

No file chosen

Fig. 8: Profile of Registered user

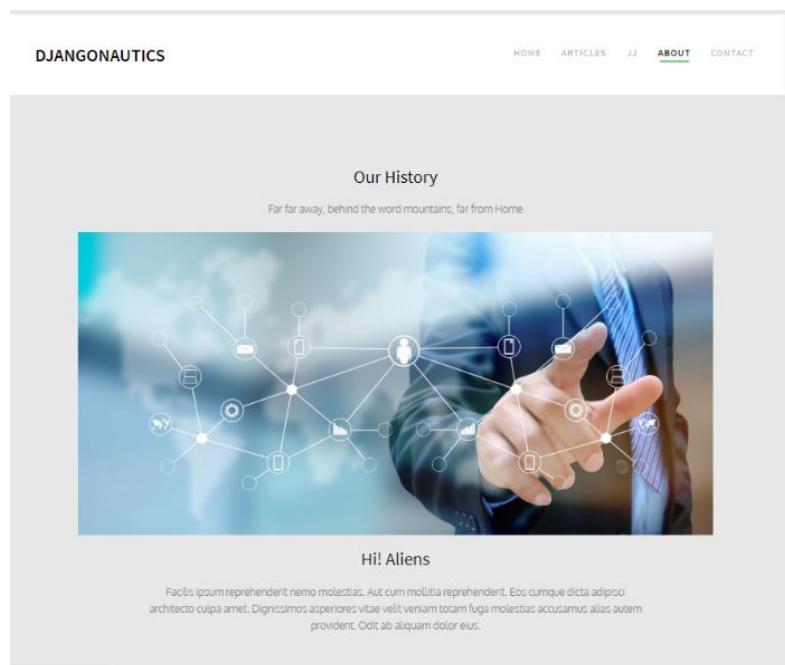


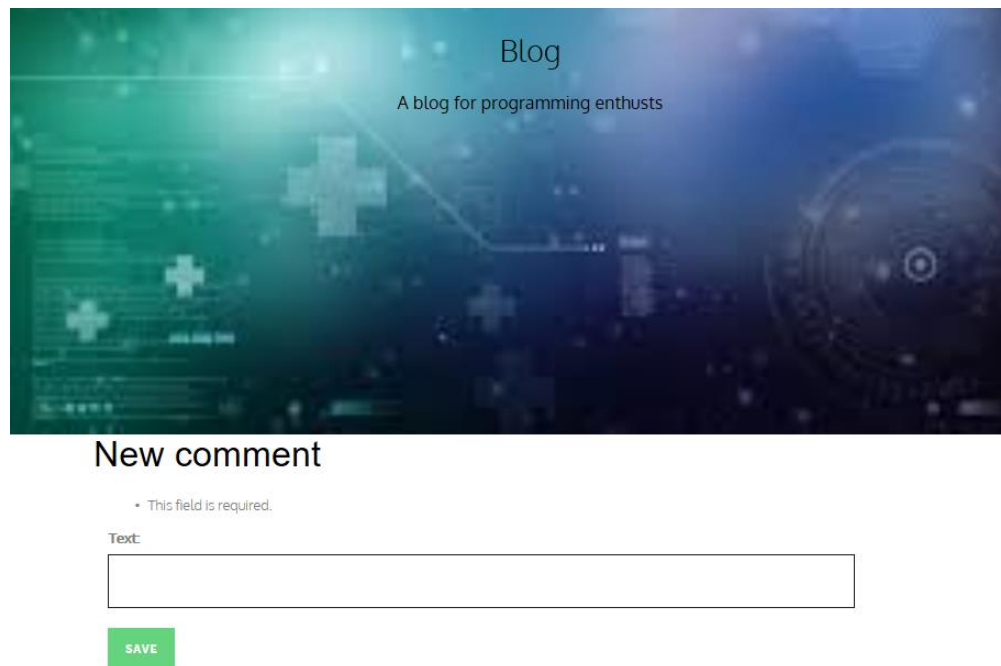
Fig. 9: About Page

## 2.6 Type of Users:

- Admin: Is the one who has control of the whole blog, the admin can give roles to users who wants to register in the blog, can change those roles according to the requests received from the users, can delete a post or modify it, admin can delete comment and user accounts.
- Author: Can post articles, modify them or delete them, comment on other post without being inspected by the admin.
- Reader or RegistredUser: Is a user who had registered and can only read articles, can comment on posts and post new article.
- Anonymous User: is a user who didn't register to the blog, however, has access to see the posts, comment on them.

## 2.7 Commenting blog post

To access the content the reader should click on the post title and will redirect to the post page displaying the whole content Fig 9. The user has the option to comment on the post using the comment option in the read post window. The comments will be displayed below each post with number of total comments. Also, in the post window there is a delete button for the author so that he can delete the post. The post can also be deleted by the admin if he found the content in the blog is licit or infringe the media rights.



The image shows a screenshot of a web application interface for a blog. At the top, there is a header with the word "Blog" and the subtitle "A blog for programming enthusiasts". Below the header, there is a large, dark, abstract graphic with green and blue hues, featuring a large white cross and some faint, glowing lines. Below the graphic, the text "New comment" is displayed. Underneath this text, there is a small red asterisk followed by the text "This field is required." and the label "Text:". Below the label is a large, empty rectangular text input field. At the bottom of the form, there is a green button with the word "SAVE" in white capital letters.

Fig. 11 Individual blog post page

## 2.8 Admin Log in

Basically, the admins are the owners of a blog that possess all rights of the blog. They can control what is being displayed in the blog and give permission to users to have a certain role. They can delete and modify the content posted by other user if the admin feels it is not in line with the ideology of the blog, and to approve or deny a comment from an anonymous user.

To be an admin, these steps can be followed to make it happen:

1. In the command window go to the root directory where the project is present.
2. Then type the command “python manage.py createsuperuser”
3. Enter your desired username and password. Once the admin is created, use the ‘python manage.py runserver’ command to develop the server connection.

There is a separate login for admin which can be accessed from the home page by clicking the admin tab which will direct to the Fig. 10 window where admin can access to the admin profile with super user rights.

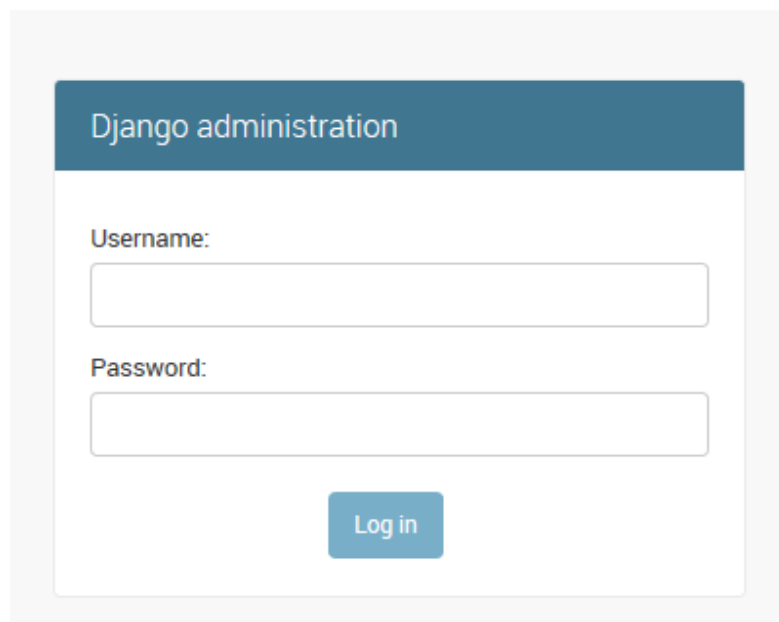
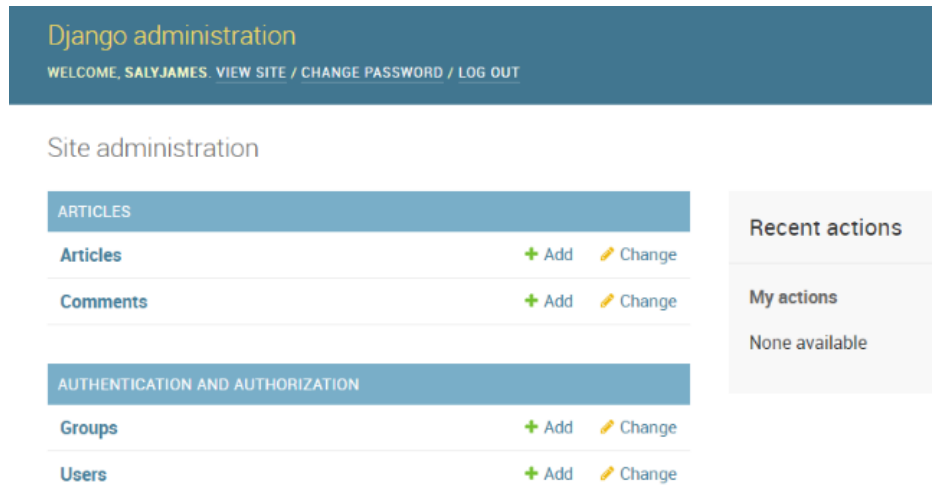
The image shows the Django administration login page. It features a dark blue header with the text "Django administration" in white. Below the header, there is a white box containing the login form. The form has two input fields: "Username:" and "Password:". Below these fields is a blue button with the text "Log in" in white.

Fig. 12 Admin login page

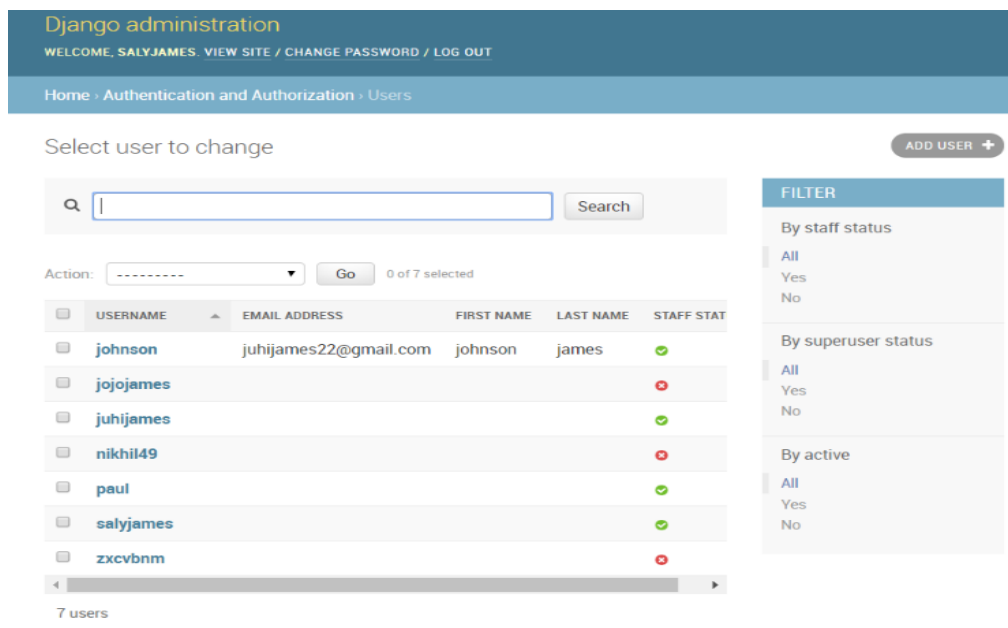
## Admin credentials:

Username – nikhil48

Password – mnbcxyl



All the apps will be displayed in admin. Here we can control all blog functionalities such as add, delete and modify the articles and comments.



## 2.9 Activity Flow Diagram

The activity flow diagram interprets the functioning of Programming World website in a simplified manner. Fig. 11 shows the UML variant of activity flow chart that describes the control flow of algorithm. Though any user can view the blog pages and comment on blog posts as per their wish, only a registered user or the admin can write or delete a blog post. If the user has account, he/she can sign in to Programming World with login credentials and if not, they can sign up to Programming World by filling the sign-up form and enjoy the blogging features.

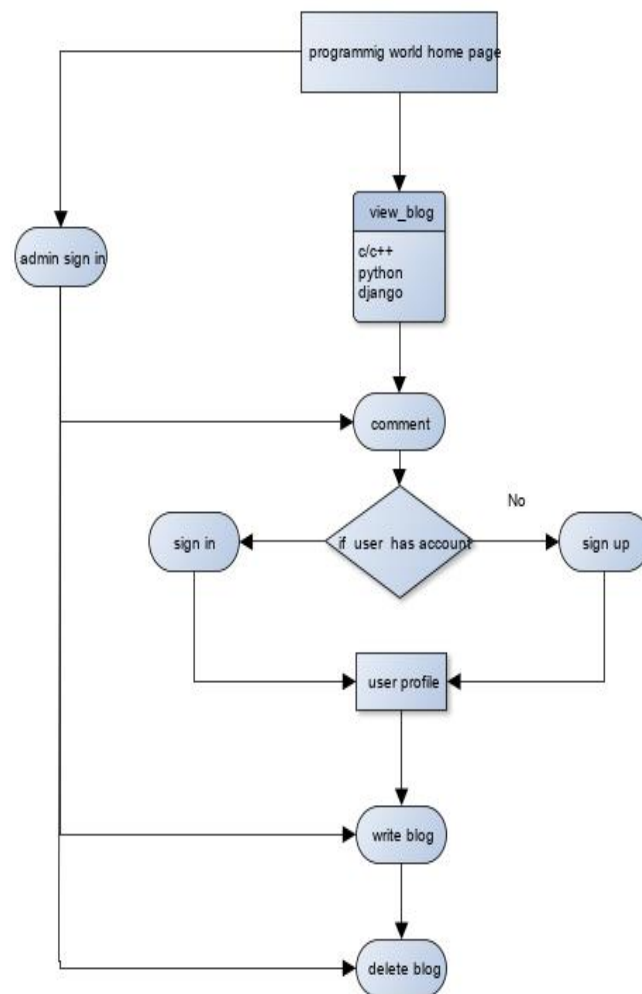


Fig. 13: Activity flow chart of Programming World

## 2.10 Use case diagram

A use case diagram is a graphic depiction of the interactions among the elements of our website. There are three actors in this website: Admin, register user and non-register user. Activity of the actor can define by using use case. For example, login, view blog, Delete a Blog etc.

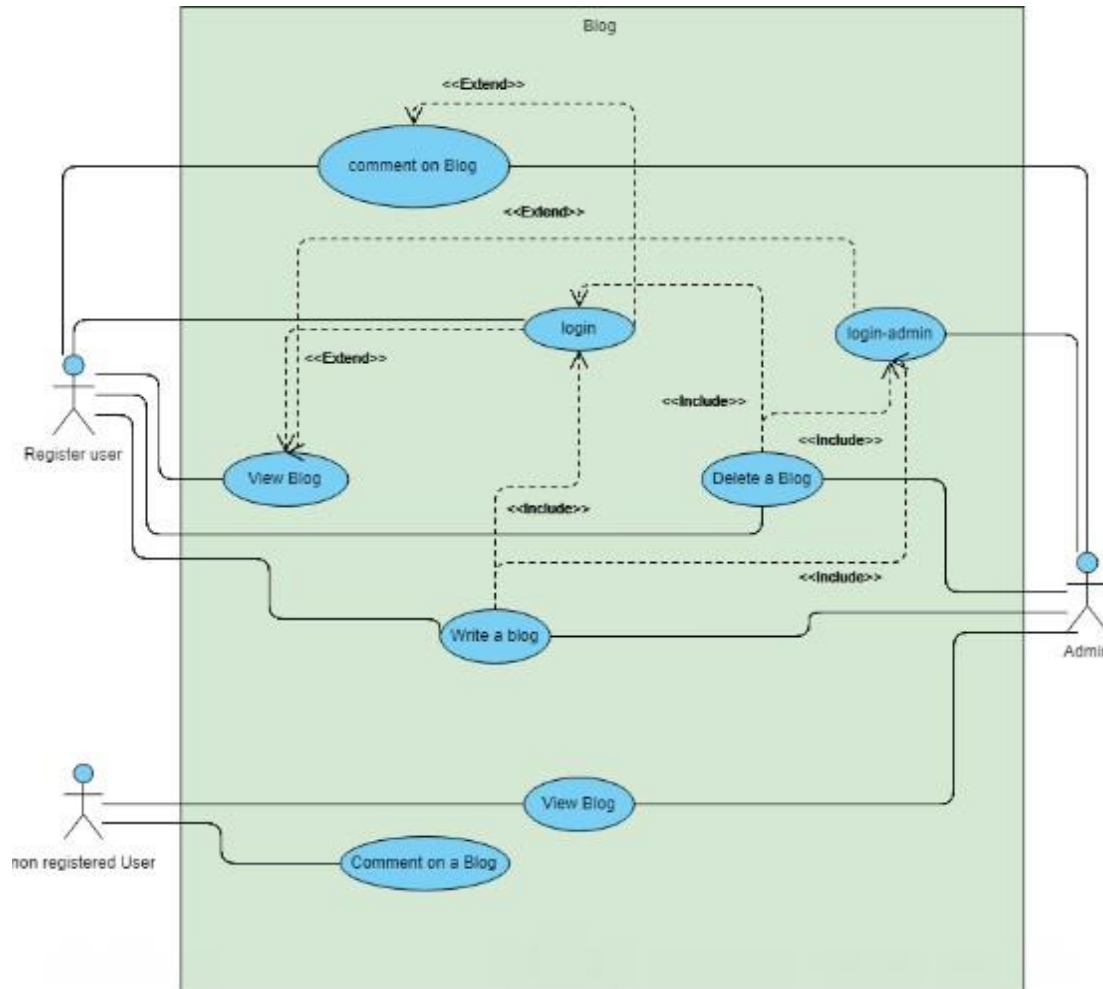


Fig.14 Use case diagram

Association (—): An actor is related to the use case when it triggers it. This relationship is represented by a connecting line between the use case and the actor. For example, register user can delete a blog. This activity can define in a use case diagram by using association between Register user and Delete a Blog.



Include ( $\rightarrow$ ): If a use case is contained in a second one, ie if it is an integral part of it, both use cases are linked with an arrow, which receives the stereotype "include" as a caption. The arrowhead points to the included application. This attribute defined conditions that must be true to use that use-case activity. For example: There are Include attribute between login and Delete a Blog as shown in above diagram, which describe that for delete a blog user must first login.



Extend ( $\leftarrow$ ) : If a use case is extended by a second under a certain condition, this relationship is indicated by the connection of the use cases with an arrow labeled with the stereotype «extend». The arrowhead points to the use case that is being extended. For example: Register user can view a blog and for that login condition is not necessary. Register user can view a blog without login also.



## 2.11 UML Class Diagram

In class diagrams, we work with the following elements:

**Class:** A class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted in our website. For example, User, User-detail, Admin, add blog, Travel blog etc.

**Attribute:** An attribute of a class represents a characteristic of a class that is of interest for the user of the website. For example, Attributes of the “Add blog” class are titleofpost, author, subject, body and



uploads.

**Operation:** In a UML class diagram, we can add operations to classes and interfaces. An operation is a method or function that can be executed by an instance of a class or interface. For example, the operations performed in an “Add Blog” are creating a blog, published date and save blog.

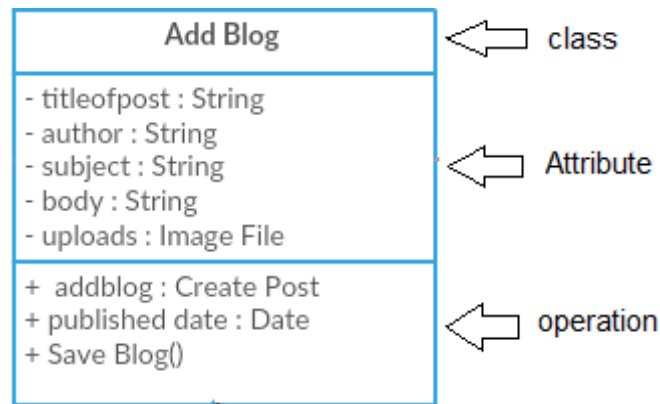


Fig. 13 Class for Add Blog

**Visibility (+ and -):** Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with (-) sign, hides information from anything outside the class partition. Public visibility, denoted with a (+) sign, allows all other classes to view the marked information. For example, in the above figure user can see all (-)attributes of the “add blog” without depending on its associated classes, therefore it shows private visibility in the diagram. Register user and admin can add a blog, therefore addblog has a public visibility.

**Generalization (→):** Generalization is a relationship between two classes. It shows strong relation. For example, Admin user and register user are subclasses of the User class (Superclass)

**Association (—):** An association indicates that objects of one class have a relationship with objects of another class. In our class diagram, objects of the admin user class and admin login/logout class have a relationship of authorization.

**Composition (Not-Shared Association):** There is a strong lifecycle dependency between the two classes. In our example, there is a composition relation between admin and add Blog class, meaning of that is when admin user delete account then add blog class is also deleted as a result. In our diagram, we can see many composition relations.

[If u want then u can add here figure also which shows above relation]

**Multiplicity:** A multiplicity allows for statements about the number of objects that are involved in

an association. In our class diagram, admin and register user can add unlimited blog, therefore we can see zero to many relations. Moreover, only one blog detail view is created by one add blog class, therefore we can see their one to one relation.

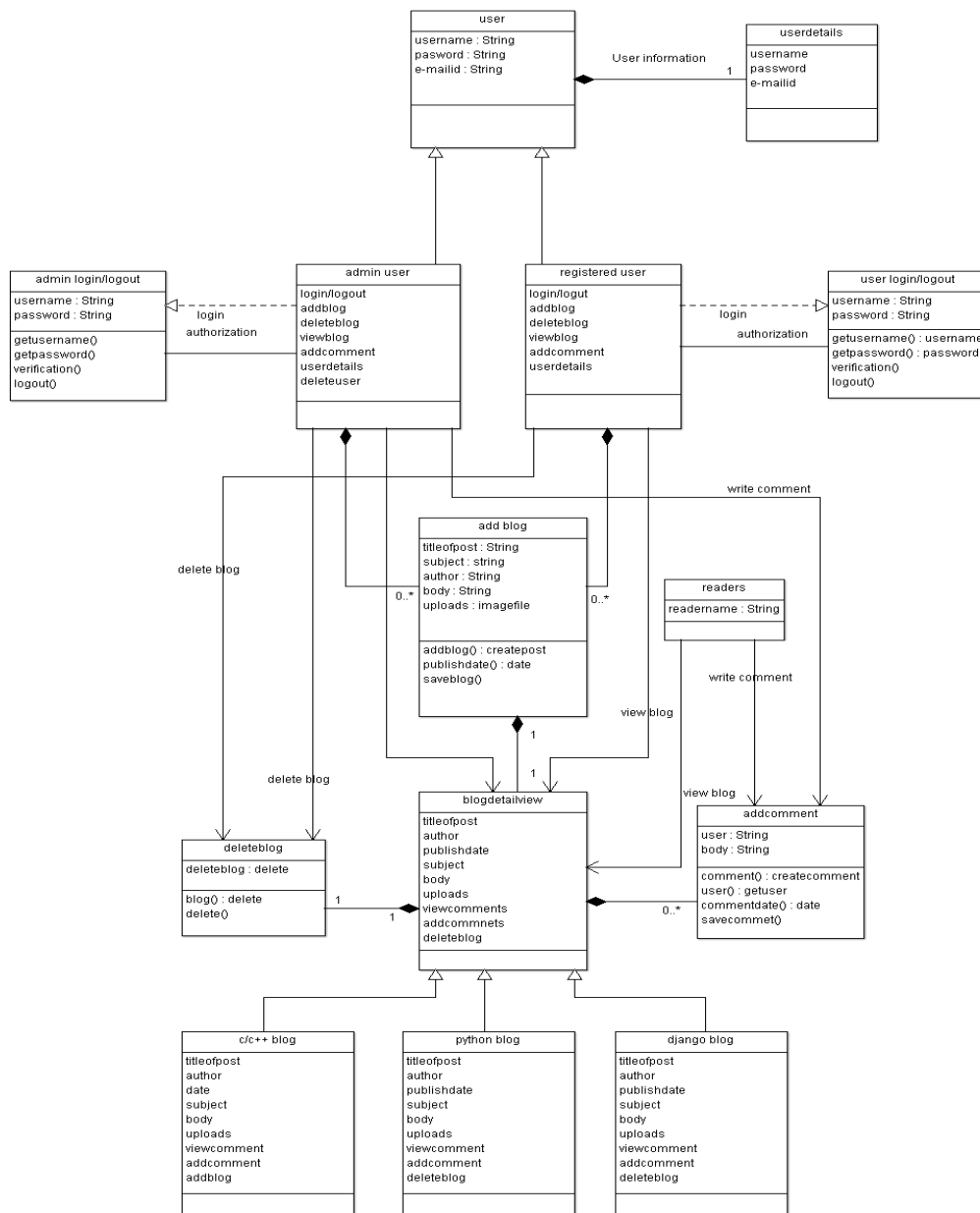


Fig 15. UML Class Diagram

Figure 14 shows the complete UML Class Diagram for our project. Functionality of this class diagram = Class user has two subclasses (admin and register user). This relation is described by Generalization.

Admin user class is associated with Admin login/logout class with dotted arrow line which implies that user successfully logged in “admin user class” by using username and password attributes of the user’s login/logout class. This same relation applies between register user and User login/logout classes.

Register user can read and delete a blog, write a comment. Therefore, we can see generalization relationship between all these classes and user-registered class. This same relation applies to the admin user class and all these classes. Reader can only read a blog and comment on a blog, so there is generalization (→) relationship between Reader class and Blog detail view class.

### 3. Code documentation

For the creation of our blog, we used Django 2.0.5 which is a web development framework that contains set of libraries and tools for creating websites and web applications. For coding purpose, we used Python as programming language. In Django, there are major pieces that need to be dealt with: Settings, Models, Forms, Views, Templates and Urls.

#### 3.1 Settings

The settings.py file contains the configuration of our website. After creating project using command: ‘python manage.py startproject finalblog’ we created our application accounts and articles’.

We have mentioned our created application under INSTALLED\_APPS.

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'articles.apps.ArticlesConfig',  
    'accounts.apps.AccountsConfig',  
]
```

## 3.2 Models

We have defined all the objects that we used for creating our Web under our application articles in file models.py. A model in Django is an object which is used to save data in database. We have stored information of users, Posts created by users and comments written by readers and uploaded images while creating blogs in database using models.py file. If we open articles/models.py it will look like:

```
from django.db import models

# Create your models here.
types_list = [
    ('python', 'Python'),
    ('c', 'C/C++'),
    ('django', 'Django'),
]

class Article(models.Model):
    author = models.CharField(max_length=200)
    types = models.CharField(max_length=6, choices=types_list, default='python')
    title = models.CharField(max_length=100)
    body = models.TextField()
    date = models.DateTimeField(auto_now_add=True)
    thumb = models.ImageField(upload_to="images/", default='default.png', blank=True)

    def __str__(self):
        return self.title

    def snippet(self):
        return self.body[:50] + '...'

class Comment(models.Model):
    post = models.ForeignKey('Article', on_delete=models.CASCADE, related_name='comments')
    author = models.CharField(max_length=200)
    text = models.CharField(max_length=400)
    objects = models.Manager()

    created_date = models.DateTimeField(auto_now_add=True)
    approved_comment = models.BooleanField(default=False)

    def approve(self):
        self.approved_comment = True
        self.save()

    def __str__(self):
        return self.text
```

In type-list we defined categories of blog such as Django, Python and C.

In this scenario of using '\_\_\_\_str\_\_\_\_' we get a text string with Post title.

We are using a SQLite database to store our data. After adding class in models, we need to perform some command to create tables in database and migrate data in database. Enter following commands to migrate data in database: 'python manage.py makemigrations' and 'python manage.py migrate'.

We can see our data available in database using command: 'python manage.py shell'.

### 3.3 Forms

We have created forms.py under our app articles to define Django's form class. We created 'Signup' and 'Add comment' functionality using form class. If we open articles\forms.py. it will look like:

```
from .models import Article, Comment
from django import forms

class PostForm(forms.ModelForm):

    class Meta:
        model = Article
        fields = ('author', 'title', 'body', 'thumb', 'types')
        widgets = {
            'title': forms.Textarea(attrs={'cols': 50, 'rows': 2}),
            'body': forms.Textarea(attrs={'cols': 130, 'rows': 10}),
            'author': forms.Textarea(attrs={'cols': 30, 'rows': 2}),
        }

|
class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        fields = ('author', 'text',)
        widgets = {
            'author': forms.Textarea(attrs={'cols': 30, 'rows': 2}),
            'text': forms.Textarea(attrs={'cols': 120, 'rows': 2}),
        }

class AnonymousCommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        author = 'Anonymous'
        fields = ('text',)

        widgets = {
            'text': forms.Textarea(attrs={'cols': 120, 'rows': 2}),
        }
```

Fields which we have defined under fields class are available on webpage to user to enter data. Entered data will be saved in database after submitting corresponding form by End-user.

### 3.4 Views

We have created view.py file under our application. It will request information from the created model and pass it to template. We have imported models, created forms and other necessary imports in view.py file. We created various functions using function (def) that takes request and return a function render which will render(put together) our mentioned template as follow: It will get data from model class and display it under template 'my\_profile\_info.html' by writing code as follow:

```
def article_detail(request, pk):
    article = get_object_or_404(Article, pk=pk)
    return render(request, 'post_detail.html', {'articles': article})

def article_each(request, types):
    limit = str(types)
    article = Article.objects.all().filter(types=limit)
    return render(request, 'articles/listofeach.html', {'article': article})
```

We added '.order\_by('-date')' under views.py to display blogs new blogs at top as follow:

```
@login_required(login_url="/accounts/login/")
def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.type = request.user
            post.save()
            return redirect('home')
    else:
        form = PostForm()
    return render(request, 'articles/article_create.html', {'form': form})
```

We used request method as 'POST' which is a Boolean value 'TRUE' or 'FALSE'. It will be TRUE if the current request from a user was performed using the HTTP "POST" method. Under

views def we saved data entered by user in form using ‘.save()’ operation as below:

```
def add_comment_to_post(request, pk):
    post = get_object_or_404(Article, pk=pk)
    if request.user.is_authenticated:
        username = request.user.username
        if request.method == "POST":
            form = CommentForm(request.POST)
            if form.is_valid():
                comment = form.save(commit=False)
                comment.post = post
                comment.author = username
                comment.save()
                return redirect('articles:detail', pk)
        else:
            form = CommentForm()
    return render(request, 'articles/add_comment_to_post.html', {'form': form})
else:
    if request.method == "POST":
        form = AnonymousCommentForm(request.POST)
        if form.is_valid():
            comment = form.save(commit=False)
            comment.post = post
            comment.author = "Anonymous-"+str(pk)
            comment.save()
            return redirect('articles:detail', pk)
    else:
        form = AnonymousCommentForm(request.POST)
    return render(request, 'articles/comment_form.html', {'form': form})
```



If we open accounts/views.py it will look like:

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth import login, logout
from django.contrib.auth import auth_view

def signup_view(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            # log the user in
            login(request, user)
            return redirect('home')
    else:
        form = UserCreationForm()
    return render(request, 'accounts/signup.html', { 'form': form })

def author_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            # log the user in
            user = form.get_user()
            auth(request, user)
            return redirect('home')
    else:
        form = AuthenticationForm()
    return render(request, 'accounts/author.html', { 'form': form })

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            # log the user in
            user = form.get_user()
            login(request, user)
            return redirect('home')
    else:
        form = AuthenticationForm()
    return render(request, 'accounts/login.html', { 'form': form })

def logout_view(request):
    # if request.method == 'POST':
    logout(request)
    return redirect('home')
```

---



### 3.5 Templates

**Django template tags** allow us to transfer Python-like things into HTML, so you can build dynamic websites faster. All the HTML files required for our projects are placed under Templates. We have used following HTML files for developing our Blog Website:

- homepage.html: Home page for our Website
- about.html: about page for our Website
- contact.html: contact information for blogger
- post\_detail.html: Complete display of single blog and all other details can be viewed
- index.html: main base of the homepage
- add\_comment\_to\_post.html: adding comment
- article\_create.html: Write blog screen for users to create blog
- comment\_form.html
- listofeach.html : list of all the articles
- signup.html: Signup screen for users to create account
- login.html: Login screen for users to login to their profile

To print a variable in Django templates, we used double curly brackets with the variable's name inside. It will display data fetched from database of that variable. Other conditional lines like for and if statements are used to add conditions to display blogs on particular webpage. <header>, <style>, <footer>, <script>, <body> such HTML tags are used in HTML files to define layout and behavior of webpage. Below code is used to display Blogs on profile webpage which are posted by logged in user only.

```

<div class="container">
  <div class="row">
    <div class="col-lg-8 col-md-10 mx-auto">
      <h2> Blog History </h2>
      <hr>
      <div>
        <ul>
          {% for post in posts %}
            <ul>
              {% if user.username == post.author %}
                <h2><a href="{% url 'post_detail' post.pk %}">{{ post.titleofpost }}</a></h2>
                <p>Posted by {{ post.author }} on {{ post.date }}</p>
                <p>Subject : {{ post.subject }}</p>
                <p>{{ post.body |truncatewords:10}}</p>
              {% endif %}
            </ul>
          {% endfor %}
        </ul>
      </div>
    </div>
  </div>
</div>

```

In login.html we have added authentication conditions for successful login as follow:

```

{% extends 'homepage.html' %} {% block content %}

  <div class="col-md-7 col-md-push-1 animate-box">
    <div class="row">
      <h2>Log in</h2>

      <div class="col-md-12">
        <div class="form-group">
          <form method="POST" class="post-form">{%
csrf_token %}

              {{ form }}
            </form>
            <p>Not got an account? <a href="{% url
'accounts:signup' %}">Sign Up</a></p>
          </div>

        </div>
      </div>

    </div>

  </div>

{% endblock %}

```

Style and Bootstrap links included in each html file as follow:

```
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Djangonautics &mdash; A blog for programming enthus</title>

    <!-- Facebook and Twitter integration -->
    <meta property="og:title" content=""/>
    <meta property="og:image" content=""/>
    <meta property="og:url" content=""/>
    <meta property="og:site_name" content=""/>
    <meta property="og:description" content=""/>
    <meta name="twitter:title" content="" />
    <meta name="twitter:image" content="" />
    <meta name="twitter:url" content="" />
    <meta name="twitter:card" content="" />

    <link href="https://fonts.googleapis.com/css?family=Oxygen:300,400"
rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?
family=Source+Sans+Pro:400,600,700" rel="stylesheet">

    <link rel="stylesheet" href="{% static 'css/animate.css'%}">
    <link rel="stylesheet" href="{% static 'css/icomoon.css'%}">
    <link rel="stylesheet" href="{% static 'css/bootstrap.css'%}">

    <link rel="stylesheet" href="{% static 'css/magnific-popup.css'%}">

    <link rel="stylesheet" href="{% static 'css/flexslider.css'%}">

    <link rel="stylesheet" href="{% static 'css/style.css'%}">

    <script src="{% static 'js/modernizr-2.6.2.min.js'%}"></script>

  </head>
  <body>
```

## 1.1 Urls

All the url patterns we created for our website are defined in templates file. All the necessary imports are mentioned at the top of the urlpatterns. If we open urls.py it will look like:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.homepage, name='home'),
    path('articles/', include('articles.urls')),
    path('accounts/', include('accounts.urls')),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]

urlpatterns += staticfiles_urlpatterns()
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

The articles urls will look like:

```
app_name = 'articles'

urlpatterns = [
    # path('create/', views.post_new, name='create'),
    path('<int:pk>/comment/', views.add_comment_to_post, name='add_comment_to_post'),
    path('<int:pk>/detail/', views.article_detail, name="detail"),
    path('<slug:types>/each/', views.article_each, name="each"),
    path('create/', views.post_new, name="create"),
]

```

The accounts urls will look like:

```
app_name = 'accounts'

urlpatterns = [
    path('signup/', views.signup_view, name="signup"),
    path('login/', views.login_view, name="login"),
    path('logout/', views.logout_view, name="logout"),
]

```

#### 4. Deviation from Plan


We have been able to achieve all requirements mentioned project plan. Therefore, we tried added some other useful features to the blog to make it easier for the users.

- We have planned to implement reCAPTCHA Design. But we couldn't implement that in login page.
- We had planned to add Tags for articles, but we couldn't implement it properly

Apart from this we successfully added below features

- Updating the user account by themselves. That is registered user can change their own username, mail id and image by update your page.

UPDATE YOUR PROFILE



JJj

JJj@gmail.com

Username:  Required: 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Email:

Image: Currently: default.png

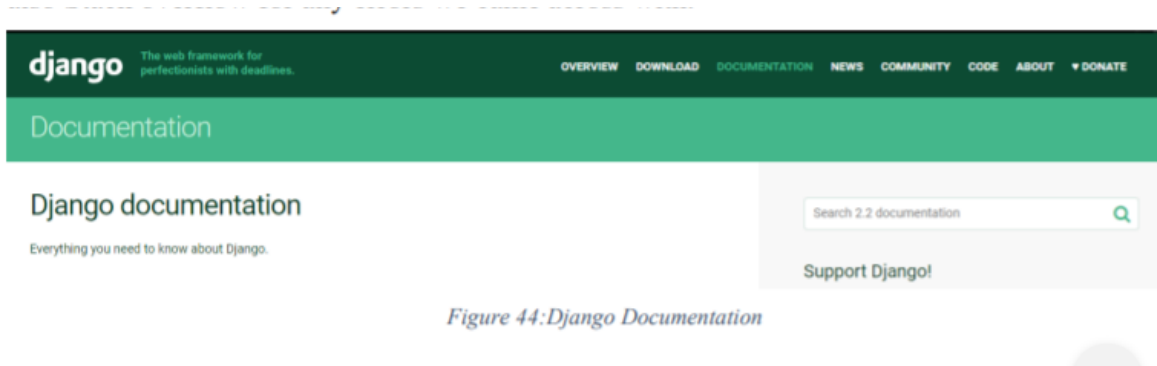
Change:

No file chosen

- Created an Article list section in the homepage. That is regardless of the category the all articles will be displayed here. Based on type of article will be displayed in the corresponding page.

## 5. Reference

We got help from many sources, one of them were the professors lectures, which introduced us to new softwares and web framework Django, the most source used was : <https://docs.djangoproject.com/en/2.2/> and <https://books.agiliq.com/projects/django-orm-cookbook/en/latest/index.html> , also Stack overflow for any errors we came across with.



*Figure 44:Django Documentation*

Figure :Django Documentation We also went for different tutorials on YouTube to get familiar with the coding using Python, html, and Css. Also Wikipedia of course, for any definition and further knowledge on the history of the software's.