Final Report

Tech Science Blog

Prepared By

1. Saif Ali 00096587
2. Mansoor Saleem 775732
3. Muhammad Ali 732065
4. Muhammad Daniyal Butt 772823

Submitted to

Prof. Andreas Fischer

Acknowledgement

I would like to Thank Professor Fischer Andreas for his expert advice and best guidance. We succeeded in our project just because of professor direction and useful tips.

I would be feel glad to thank all of our group members.

Table of Contents

Chapter 1

Tech Science Blog

Introduction:

A blog is an online diary or journal located on a website. The content of a blog typically includes text, pictures, videos, animated GIFs and even scans from old physical offline diaries or journals and other hard copy documents. Since a blog can exist merely for personal use, sharing information with an exclusive group or to engage the public, a blog owner can set their blog for private or public access. Our blog website is a site that is updated with new information on an ongoing basis of sports. It normally consists of a collection of Sport posts. “myproject1” is a Sport discussion Posts are typically displayed in reverse chronological order, so that the most recent post appears first, at the top of the web page we can also factorization using the Ascending and Descending list button. After all user or author must be signup there then he/she can read, Edit, upload, and Delete the Blog.

Our sports blog needs to stand out in a way that makes it easy for visitors to digest every bit of information within the first few seconds. They should have a good idea of what they’re in for once they land on your site. Keeping up to date with today’s trends for sports websites, here’s how you can make your web presence really stand out. We develop a as per user usability UI for more attractive and Increasing quantity of blog traffic. User can easily contact us to using link of contact form and share with us all feedback and complains. On the Blog homepage tag is linked with Articles user can approach easily. Another added benefit of blogging is that it helps playing sports is a generally a fantastic way to improve your fitness and health. Many of us may not feel at home pounding away on a treadmill or working up a sweat in the gym, but we’ll happily chase a ball around endlessly while playing a game of some sort. A sports blog can accomplish a number of things. It can help the team stay updated with the upcoming events, attract new players and make it easy for them to contact you, or help fans stay in touch and get updated on the latest news. Most importantly, it can help the sports team market themselves better by having an online presence.

For most people, taking part in sport will improve your general health and wellbeing. There are plenty of reasons why you should become involved in sport with reduced body fat, bone strengthening, improved stamina and flexibility being some of the reasons why you should take up a sport. Through sports you will meet people with a similar interest to yourself and are likely to gain many new friends. Sports allow you to challenge yourself and set goals.

Aim:

All blogs need goals. Without goals, you won't accomplish anything at all. But our main purpose blog it helps out to keep update with the matches and incoming new players and keep heath life. We develop the blog using Python Django framework, HTML and CSS.

1. **Improved cardiovascular health.** The heart is a muscle, it needs to be worked out! Regular exercise can help improve the overall health of your entire cardiovascular system.
2. **Lowers risk of heart disease, stroke, and diabetes.** A healthier heart means reduced risk of cardiovascular disease, stroke, and diabetes.
3. **Helps manage weight.** Not only does physical activity burn calories, it also improves your metabolism in the long run.
4. **Reduced blood pressure.** Physical activity keeps your heart and blood vessels healthy, helping to prevent hypertension.
5. **Improved joint flexibility and range of motion.** Improved flexibility reduces risk of injury.
6. **Stress relief.** Exercise is a great mood-booster and has proven to be an effective method of stress relief.
7. **Lowers risk of certain types of cancer.** People who exercise regularly are less likely to develop breast, colon, and lung cancer.
8. **Control cholesterol.** Exercise decreases LDL (bad cholesterol) levels and increases HDL (good cholesterol) levels.
9. **Improved sleep.** We know just how important sleep is, and exercising can help you capitalize on these benefits.
10. **Mental health benefits.** Exercise is good for your mental health too, as it can battle feelings of anxiety and depression, sharpen your focus, and improve self-esteem.
11. **Prolonged life.** When you add all these benefits together, what do you get? A longer, healthier, more enjoyable life!

Objectives:

A sports blog can accomplish a number of things. It can help the team stay updated with the upcoming events, attract new players and make it easy for them to contact you, or help fans stay in touch and get updated on the latest news.

- Keep the Visitors Updated with News and Blogs.
- Current Team Roster, A sports team is dynamic, and there are changes happening all the time due to various reasons including a new player, a retirement or even injuries.
- Schedules are an important and very easy way to make sure everyone associated with the team knows about any upcoming events.
- In this day and age, and it's almost expected that your organization is on the web in some shape or form.
- Displaying leader boards and top scorers on your sports blog is also a great way to encourage competition amongst your team members to get them inspired and motivated to aim higher.
- Every sport Articles needs a blog section to help visitors know what's going on with the club or sports.
- Every true fan and sports organization would want to keep in touch with through our blog. As far as sponsorship offers and press inquiries go, you should be making it extremely easy for people to connect with your sports blog
- Have detailed team and player stats displayed on our blog so that visitors can see exactly who is at the top of their game.

Chapter 2

Structure of Project

The BLOG project here is being made by using the framework of PYTHON called DJANGO. Now the important thing is to install django itself which is nothing but a lib. We did by entering the following command in powershell.

“pip install django”

The first thing we did is to create a directory in the desktop named “myproject1” and then we entered into the path into the power shell and entered the following command to start our django project

“django-admin startproject myproject1”

By this command the directory gets django related directory. Now it contains some python files like manage.py,urls.py,views.py,and settings.py. Now when we run our server at first by “python manage.py runserver “ command in the promptshell , our server runs at the local host of http://127.0.0.1:8000/ but as we ran our program at first it had some unapplied migrations for which we had to apply migrations by writing “python manage.py makemigrations”. Now the basic concept that is being used throughout in our program is the concept of how browser interacts with the backend .Well a user makes a request to the browser which then passes to the urls.py file in django , now this file looks at the requested URL by the user and gets connected with another file called views.py file in the same directory , All functions are made in views.py and so our urls.py file decides whether which function of views.py to fire for the request made by the user. So at the end views.py actually sends the response back to the user on browser. Now we imported the following things into the urls.py (myproject1/myproject/urls.py) of our project

from django.conf.urls import url

from django.contrib import admin

from . import views

then in urlpatterns we declared some urls , starting with the raw string “r” inside ‘ ’ we wrote the carrot ^ first after which we the name of slug that we want our site to show in the url of website. We used admin , homepage and about , in urlspattern . To end the url we put $ sign. We made the functions of each and everyurl that we wrote in the urlpatterns. For example for homepage and about we made two functions into our views.py and imported each and everything written in the views.py. Now each and everyurl has some string part followed by views.”name of function made in views.py” for example for “about” function the url is

url(r'^about/$', views.about,name="about"),

2.1 TEMPLATES:

For the front end templates we made the folder named templates as

“myproject1/templates”

Now we stored the name of templates in settings.py files under the “TEMPLATES” and inside the “DIRS” .i.e.

'DIRS': ['templates']

Django Apps:

“articles” app:

Instead of writing each and everything in the same path, we used a technique of creating apps , now for different tasks we divided the process into different apps , and each app has its own urls.py , views.py and templates etc. Since our Blog is all about articles so we started our first app by the name of “articles” by the command in command prompt below and also we entered the name of our app in settings.py under INSTALLED APPS.

“python manage.py startapp articles”

Urls.py: Now in articles app we created a python file named urls.py and did the same stuff described above in the report. The url that entered here has no raw string and fires a function named article_list defined in the views.py as

url(r'^$', views.article_list, name="List")

we included the same url into the urls.py file in our main directory (myproject1) using “include ”command

Views.py: article_list is the function is the first function being made in the views.py which simply renders a template named article_list.html to the user.

Article_list.html: for name spacing we created a folder named templates inside our app articles , and inside templates we created a further directory named articles where we created our html named article_list.html. Our article list contains a search bar with <form> tag which searches the article on the basis of its name, it contains a for loop for the tags of article as well as another for loop for displaying each and every article along with the name of article , 50 characters body ,the date when its being published ,the name of the author as well as the picture as a teaser if an article has image. In order to serve different functions our article_list.html has different urls .And the concept is simply to use template tags, for processing we used{{%%}} and for output we used {{}} as tags.

2.2 Models of Article:

Models in python or django are a class which represent a table in a database, now different type of data is represented by its own model and each model is being mapped into a single table in a database so generally:

Each model is a Python class that subclasses the "django.db.models.Model".

Each attribute of the model represents a database field of its own.

With all of this, Django gives you an automatically-generated database-access.

The model that we created lies in the models.py file of directory called Article which has parameters of models imported from django.db. Each and every model has its fields which represent that what are the things present in the table. The fields that we have set for our model are

1. Title: for which we have used CharField , field type with the maximum length of 100 words.
2. Slug: Which has field of SlugField, mention in django documentation.
3. Body: Since body contains large number of strings that's why its field type is set to TextField.
4. Date: Which has field of DateFieldTime and a parameter which causes to store the date when an article is being written.
5. Thumb: which is an ImageField, to store pics.
6. Author : which has been inserted by using a Foreign key , and has parameter of the User.

There are two functions being added below __str__ which returns the name or the title of the object saved in database and the snipped function which simply restrains the number of string letters to 50 words that we used in article teasers.

MIGRATIONS:

Migrations are Django's way of propagating the changes that we make to our models (adding a field, deleting a model, etc.) into your database schema. As we created a model named Article we certainly need migrations to map in into a table and that purpose is being served just by the following commad in prompt: "python manage.py makemigrations"

Then it also necessary to migrate those changes into the workspace for which we type:

"python manage.py migrate"

The Django web framework includes a default object-relational mapping layer (ORM) that can be used to interact with application data from various relational databases such as SQLite3 that is in django.

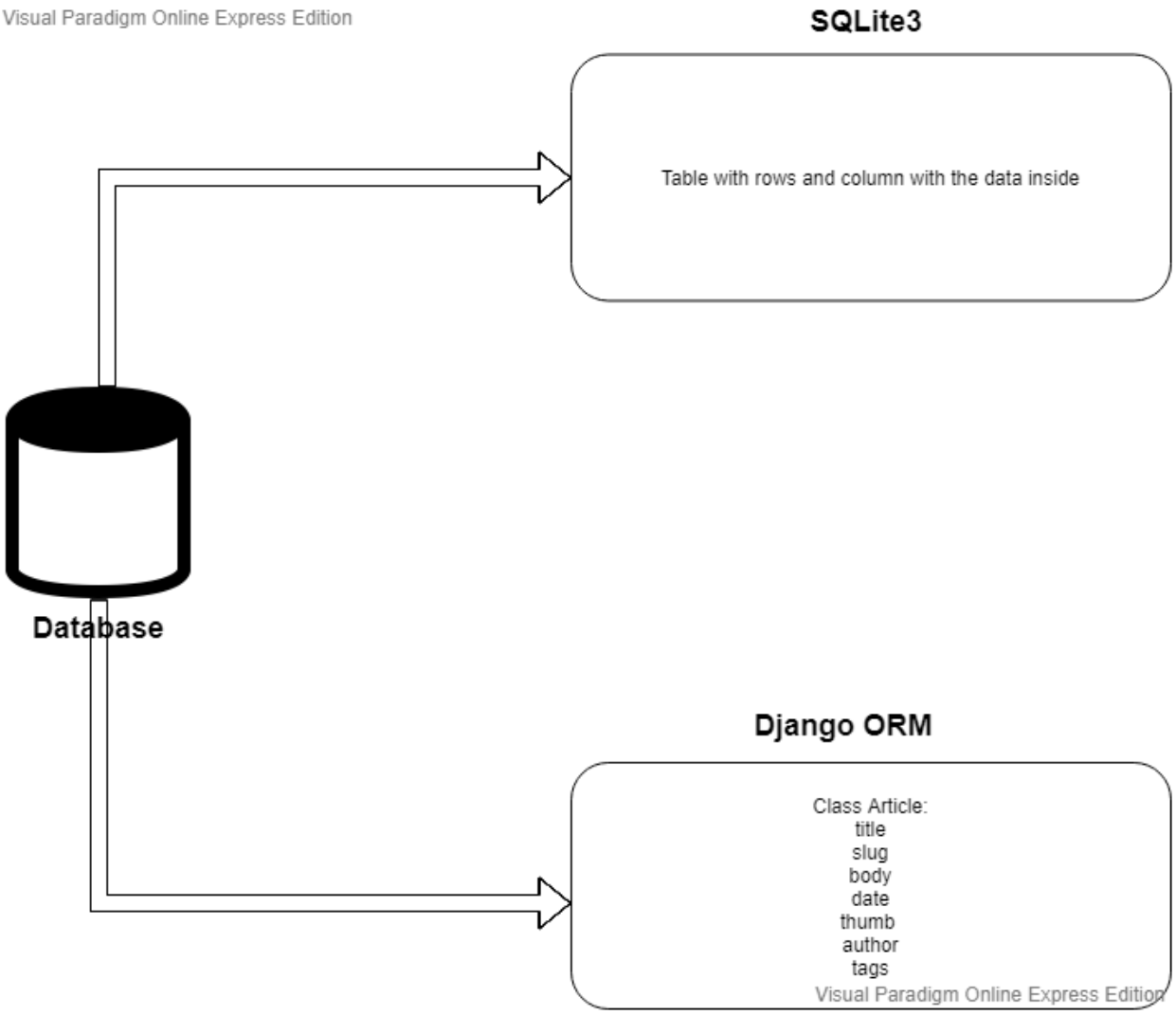

Figure 1Database framework

Chapter 3

Admin

3.1How to create Admin:

Djangoadmin area which is a really great feature of Django comes fully baked withitdeveloper don't have to do anything to setit up, so the adminarea is an area for site admins tocontrol the content of the websiteessentially or the contents of databaseswe can create instances of models suchas articles. we cancontrol users of our website etc. After started this project we have this URLs file in the Django app that we have this admin URL set upwhich is

```
urlpatterns = [
url(r'^admin/', admin.site.urls),
]
```

thatsuggests to us that the admin section isgoing to be all right hereso the first thing we want to do is makesure to create a super user and run the server.

We created a superuser account to get access the admin panel. By typing the command in cmd.

```
Python manage.py createsuperuser
```

Now running the server.

```
python manage.py runserver
```

once that'srunning open a browserandpaste this in address bar

```
http://127.0.0.1:8000/admin
```

We will have admin login form where we can login and gain access of the account. we can see groups and users. By clicking users we can see all users including superusers. We can see Article table where we can see articles and comments section in admin page which we added instances of articles models. Articles model is appearing in the admin page because we registered it in articles app folders admin.py file. For this we had to import few model->articles. if we click on article we will see all articles titles which are posted and have add article button where admin can add articles to post. After clicking one of the article titles we can see article form where we have ability to edit and delete the article. If admin click on comments, then a page will appear in

front of admin where admin can see a form which contain Post, Author, Text, Created date and time. Admin have ability to edit and delete the comment also.
Admin will have also tag section on his admin page. Clicking on tag admin can see another webpage where he can see a table with two columns. One consists of tags and second with slug. And admin have ability to add tags also.

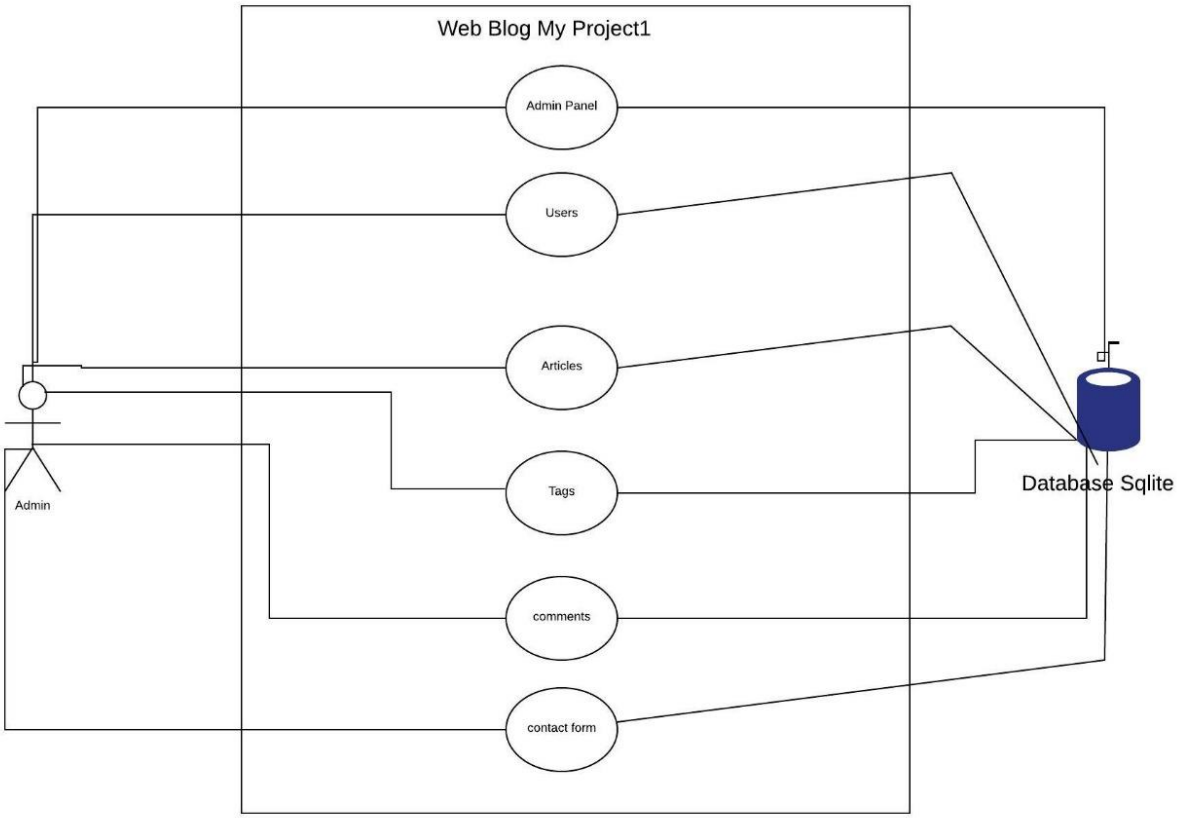

Figure 2 Admin role

Chapter 4

Accounts

4.1 LOGIN and SIGN UP:

Django comes baked with user authentication system by which we can create a registered user and get access to the extra features provided by the website. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the report explains how the default implementation works out of the box, as well as how we extended and customized it to suit our project's needs.

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

The authentication system consists of:

1. Users

2. Permissions: Whether a user may perform a certain task.

3. Groups: A generic way of applying labels and permissions to more than one user.

4. A configurable password hashing system

5. Forms and view tools for logging in users, or restricting content

6. A pluggable backend system

The authentication system in Django aims to be very generic and doesn't provide some features commonly found in web authentication systems. Solutions for some of these common problems have been implemented in third-party packages:

1. Password strength checking
2. Throttling of login attempts
3. Authentication against third-parties
4. Object-level permissions

4.2 INSTALLATION OF “ACCOUNTS” APP:

Firstly, in our program we started an app called “accounts” for the sake of users getting registered by the following command:

“python manage.py startapp accounts”

Later on we add the similar “accounts” app into the settings.py file under the heading of installed apps.

Authentication support is bundled as a Django contrib module in django.contrib.auth. By default, the required configuration is already included in the settings.py generated by django-admin startproject, these consist of two items listed in your INSTALLED_APPS setting:

'django.contrib.auth' contains the core of the authentication framework, and its default models. 'Django.contrib.contenttypes' is the Django content type system, which allows permissions to be associated with models you create.

and these items in your MIDDLEWARE setting:

SessionMiddleware manages sessions across requests.

AuthenticationMiddleware associates users with requests using sessions.

With these settings in place, running the command manage.py migrate creates the necessary database tables for auth related models and permissions for any models defined in your installed apps.

Usage of django built in libraries:

Using Django’s default implementation.

1. Working with User objects
2. Permissions and authorization
3. Authentication in web requests
4. Managing users in the admin
5. Customizing Users and authentication
6. Password management in Django

4.3 The Baked in Django auth app:

Django automatically installs the auth app when a new project is created. Look in the settings.py under INSTALLED_APPS and you can see auth is one of several built-in apps Django has installed for us.That is present in “myproject1/settings.py ” under INSTALLED_APPS

Urls.py:

At first in the urls.py we imported the urls from django.conf.urls . Since django.conf.urls is an Helper function to return a URL pattern for serving files in debug mode. Then for the next one we simply imported views from the same directory where we are created our functions for the sake of Login and Logout. To design URLs for an app, we created a Python module informally called a URLconf (URL configuration). This module is pure Python code and is a mapping between URL path expressions to Python functions (the function views that we created). The urls started with the raw string , the carrot and then the slug being displayed with the dollar sign to end the url.After the address we included the views functions i.esignup_view, login_view and the logout_view . The Urls that we created inside our url patterns are 3 named:

1. signup
2. login
3. logout

Creating the functions in views.py:

First of all the things that we imported so far that we used as for creating our functions are :

1. from django.shortcuts import render, redirect
2. from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
3. from django.contrib.auth import login, logout

the same django.contrib.auth.forms comes into handy when try to take the advantage from the django built in forms such as UserCreation Form for the creation of form as per sign up and the AuthenticationForm to confirm the user and authenticate him to the desired urls.

Then django.contrib.auth we got the two basic functionalities that are also ready-mate in django named the login and login, along with two commands from django.shortcuts , render and redirect , to fire a function or to redirect to a certain function respectively.

Firstly we created the signup function in view. Which has an If-Else statement. The parameter inside all of our functions so far is that of request since the functions always gets need to have a request

Now in our “signup_view” function we used an approach inside our If-Else statement that is

1. The request method could either be POST or GET but when a person is trying to log in or sign up the page, the approach he uses is that of POST. So we declared that if the method is to POST
2. Then store the entered data into a variable called form which implements the UserCreationForm built in function and takes the request which should be that of POST.
3. Now another “IF” statement being used along with django property “.is_valid” to check whether the form is valid i.e. the passwords match? or the username is correct so and so on.
4. So when this condition is satisfied, the next statement that is being executed is “login” with parameters of request and the user that was trying to fill the form.
5. As the Form was valid and now user is logged in with the credentials , we fired a return function which redirects to articles list (where all the articles are displayed)
6. But what if the form that the user filled is not valid , well for that case we do have an else statement which stores nothing but instead executes a return function rendering the request and sending the same user to sign up html template which will be described in the next page

The second function that comes into handy is the “login_view” function which has kind of the same approach as that of “signup_view” with some noticeable differences , well this function also uses IF-Else statement as follows:

1. The request method here is also to POST since a person here is trying to enter the saved data to have access to the new features which only a registered user can have.
2. Secondly the form variable stores the data which has POST type through a django built in lib called AuthenticationForm which we imported up on the code from django.contrib.auth.forms
3. Now similarly as above another If-Else statement is being used to check whether the entered data is valid or not by previously used “.is_valid”
4. Then the changed statement comes compared to that of the signup form which is that the user variable stores the form and the get user data that is present into the data base
5. After which it follows the same “login ” command to save the user request.
6. This part becomes crucial now, since instead of sending the user back to the articles list we can check whether the “next” property exists in the post data, now if it does exist we can send them to that value so we used an if statement in the program to see that “next” property exists in request.POST
7. Then redirect to the request GET to the parameter in the post which is “next ” which is mentioned in the html with the input template named “next”
8. Else if we don’t get that redirect us to the articles list

9. But what if the login credentials don't match with any user or the form is simply incorrect , well then the Else statement of the first loop comes into play which simply stores nothing in the form variable and then renders back to the login template which would be described below

Well now the last and the final function in views.py is our "logout_view" which is simply the most simple function so far , the logic is that of an IF statement

1. Check if the method is that of a POST
2. If it is of POST type, simply use a built in function called logout, which is imported from the djanogo.contrib.auth as similar to that of login used above.
3. So the logout function having a request as a parameter logs the user out and then the last command which simply fires a return function redirecting the user to the article list page.

Well these are the processes that are happening in the background of our back end three functions that have the responsibility to store the correct or the right credentials into the login or signup form having the method of POST or to log a certain user out of his workspace respectively. The concept of "next" is important here since what we like is not to refer or render always a user that gets logged in into the page to article list but to the next parameter which if is available would get user there.

4.4 RESPECTIVE TEMPLATES:

The two templates that are used here are the login.html and the signup.html. The location of these templates are "accounts/templates/accounts"

Signup.html:

The signup template first starts with extending the "base_layout.html" through a template tag {% extends 'base_layout.html' %} which kind of inherits the baselayout properties . The next one is the content that we have to write , so we use another template tag here which contains our main content. It starts with {% block content %} and ends with {% endblock %}.Inside this is our main functionality which first of all starts with an heading of <h1> tag named "Signup" the second thing being the form which is the most vital or the core thing of our signup html.

And finally our form that we created in the views function, but here we have displayed the form using template tag {{ form }} . Which simply displays the form which we declared in "signup_view" in views.py. And at the end we simply need input tag (<input>) of submit type and having the value of Signup . To execute or to fire the entered data into the form.

4.5 Login.html:

The concept used in login.html is more or less the same that used in signup.html other than some differences. It starts with extending the base_layout.html and then the block content template tag starts. Inside the block content goes in the heading <h1> tag first displaying "Login" , then a

form having a class of “site-form” which performs an execution of url in accounts named “login” having a method of post .Then comes our security token {% csrf_token %}. And here comes our {{form}} which is being displayed into the template ,that particular form which was in login_view function in views.py., so we again used an if statement template tag {% if request.GET.next %} to see whether the request is to GET next parameter If It is so then execute an input tag <input> of type hidden , name next and value {{ request.GET.next }}. Then we end our if statement by tag {% endif %}. And then to submit the login we used input tag with type of submit and value of login.

Following is the sequence diagram of sign up and sign in.

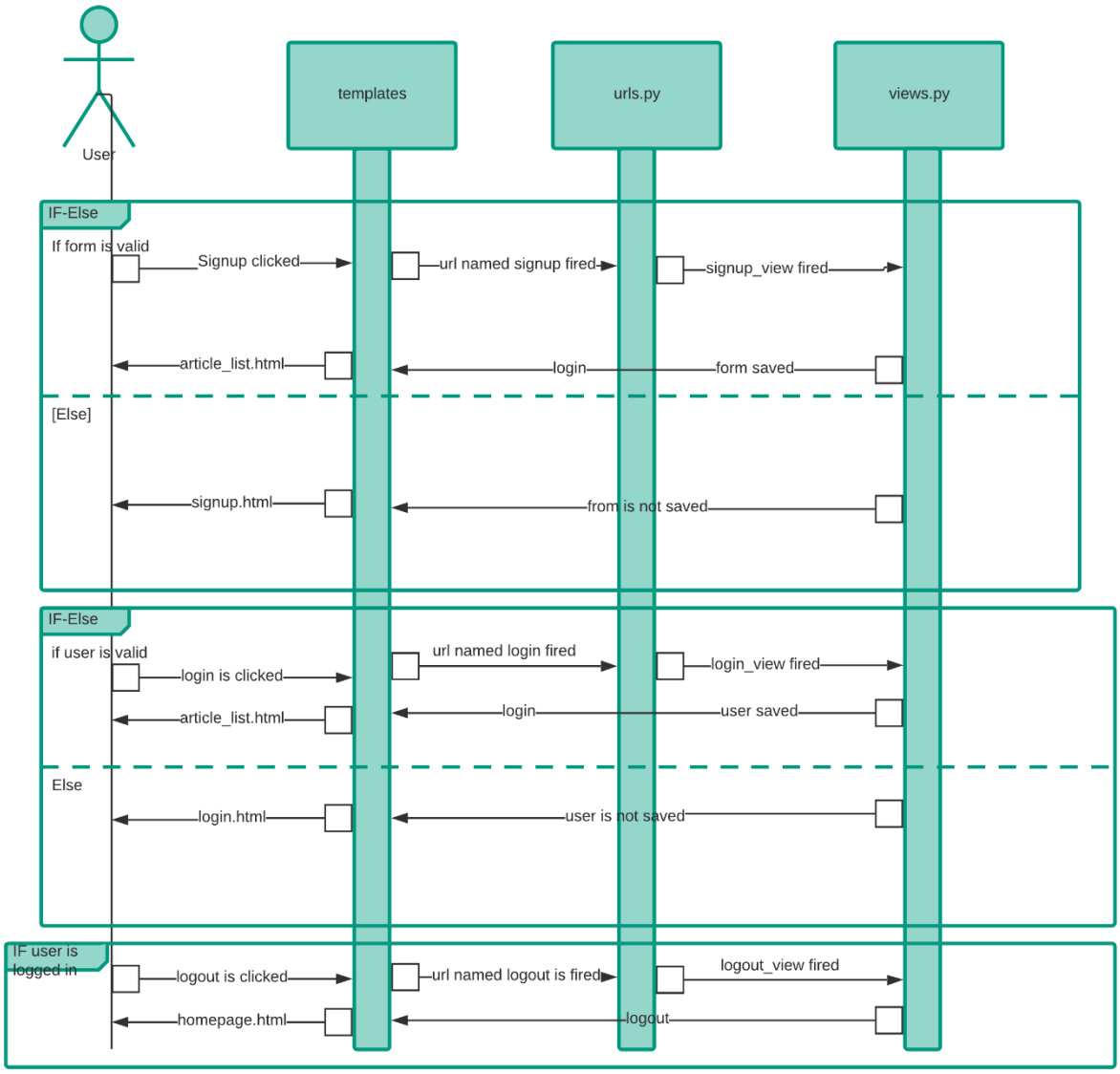

Figure 3 Accounts

Chapter 5

Article

5.1 Static files

Configuring static files¶

1. Make sure that **django.contrib.staticfiles** is included in your **INSTALLED_APPS**.
2. In your settings file, define **STATIC_URL**, for example:

```
3. STATIC_URL='/static/'
```

4. In your templates, use the **static** template tag to build the URL for the given relative path using the configured **STATICFILES_STORAGE**.

```
5. {%loadstatic%}

6. <imgsrc="{%static"my_app/example.jpg"%}"alt="My image">
```

7. Store your static files in a folder called **static** in your app. For example **my_app/static/my_app/example.jpg**.

Serving static files during development¶

If you use **django.contrib.staticfiles** as explained above, **runserver** will do this automatically when **DEBUG** is set to **True**. If you don't have **django.contrib.staticfiles** in **INSTALLED_APPS**, you can still manually serve static files using the **django.views.static.serve()** view.

static files are things like images CSS files or JavaScript files that we can serve up to the client the browser now in Django it's not as simple as in this html template we want to create an image so here's the image tag and this is the path to the image which is stored in some kind of folder over here that's not going to work we have to explicitly set static images up inside our project and in our case we're going to tell Django to server our images now when we are creating a website for production we typically wouldn't let Django server up our images we I'm going to call this assets but you can call it whatever you want so this right here is saying okay where the static files is going to be found in the base directory then look in the assets folder so if we save this now and create an assets folder over here then we can store all of our static files inside here so for example if I say styles ss that I can now serve this up to the browser so for example body and then the background will be for now let's just do blue save it if I now go to forward slash static forward slash styles then Django is going to know how to serve up this file and I get the

CSS file back in the browser cool right so now we have this Styles dot CSS static file but how do we connect it to our template right here the articles template well I guess what we could do is say well link rel is equal to stylesheet and then the href is going to be forward slash static forward slash styles dot CSS and this will absolutely work and I can demonstrate that by going back over here and refreshing then we get the background of blue but I don't think that this is the best way to do it by hard coding the static file URL and the reason why is as follows if at some point we want to change this thing down here its assets what are something else or to a longer path for whatever reason then we will have to go into every template where we've used an asset of any kind and change the URL to update this now that would be a bollock and we don't want to do that so a better method to using static files inside your templates is to use something that comes baked with Django and we can load that right here by using our templates tags and that is called the static module so we'll say load static from static files so right here we're saying we're loading this static thing which we can use now inside this template to form our static URLs so that if we do ever change that URL in the settings then this thing right here will work it out for is we don't need to go around every template and start to reach ange all of the H ref attributes all the source attribute it's all taking care for us so in here instead of hard-coding now what we do is output our template tags and inside we said we want to use a static URL then in single quotes so that we don't escape the double quotes of the href attribute we want to say which static file we'd like to load here now I want to load in Styles dot CSS so if I save this now that hopefully fingers crossed this should still work and it does we still get that CSS file but this is a much better way of loading in those static files than to hard-code the URL right here ok then so let's add to this CSS file why not so let's close articles and open up this dude and instead of you watching me cold all of this CSS I'm just going to copy and paste from my github repository and you can find the styles.css file as well on that repo if you want to use it so some dead simple rules we have the body just declaring the background and it's actually using a static file right here and we'll go through this in a minute then every element within the body will have this font family we're starting the headers and also the links right here as well so some really basic styles to begin with but if I save this and refresh over here then we get this kind of display in the browser now right now we don't have this stars dot PNG file inside the assets folder so it's not given us that background image but I do have this in a folder over here and again this is on the github repository for this course as well so you can find that up there so I can save this now and refresh now we should get those stars in the background we can just about see those however right here again we're hard coding to begin with static and again what if that URL structure changes in the future well a better way to do this would be to just use a relative path so we can just say Stars dot PNG because that's in the same directory as the Styles okay so let's save that and this should hopefully still work yep it does ok so there we go my friends that is how we can use static files in Django and we will be using them more as we go through the rest of this series

5.2 Article Detail:

Django provide several class based generic views to accomplish task and one of them is Detail view of an Article.

Views.py:

DetailView should be used when you want to see the details of single model instance.

For DetailView firstly, we worked in view.py as following:

```
from django.views.generic import DetailView, ListView
```

and then define the function article_detail, In Article detail we used get object method and then added comment function's statement.
By adding request method POST, requested for comment form after this user can give some text in comment form and it will be saved.

Urls.py:

```
from django.conf.urls import url
from . import views
```

and then added a line in app articles url's:

```
url(r'^(?P<slug>[-\w]+)/$', views.article_detail, name="detail"),
```

Article_detail. Html:

After defining and giving urls, worked on html to make body of article detail. How should it look like and what parameters should be displayed?
In Article Detail view user can read the whole article with date and author name. Article can be commented by user. User can see the tags given by Author. And This html also connected with base layout.html of project.

5.3 Article Create:

We start working on articles by creating an application by command line:

```
python manage.py startapp articles
```

Then in Installed_apps we added 'articles'

Models.py

Then we created model Article as each model maps to to a single table in database and in model we gave all the parameters like title, slug, body, and date.

After creating model, we have to make migration and we gave command

python manage.py makemigrations
&

python manage.py migrate
after logging in by administration a datebase for article is created and admin can write and add various articles in database.

With all of these we worked on its CSS in styles.css file.

To create or write a new article, must be log in or signup on the website. Created this page by following steps:

Forms.py

First created a forms.py file and then make a new class with name CreateArticle in this and then We need to import Django forms first (from django import forms) and our Article model (from .models import Article).
ArticleForm, as you probably suspect, is the name of our form. We need to tell Django that this form is a ModelForm (so Django will do some magic for us) – forms.ModelForm is responsible _or that.
Next, we have class Meta, where we tell Django which model should be used to create this form (model = Article).
Finally, we can say which field(s) should end up in our form. In this scenario we want only title and text to be exposed – author should be the person who is currently logged in (you!) and created_date should be automatically set when we create an article.

Views.py

Furthermore, we worked in views.py and import from .import forms and def article_create function in login function

Article create html:

After this created an article_create.html
In this extends base layout template and create body for article create form.

Urls.py:

We open articles/urls.py in the code editor and then added a line:

url(**r'^create/$'**, views.article_create, name=**"create"**),

5.4 Article Edit:

As articles has been written by authors, meanwhile author has an option to edit the article. For this he just has log in and click on edit button and he will be directed to the edit page.

How Django works in article edit it is stated as follows:

Views.py:

In views.py file we defined the article_edit function by post and get object method

If request.method = “POST”

Then we will get a form where we can \

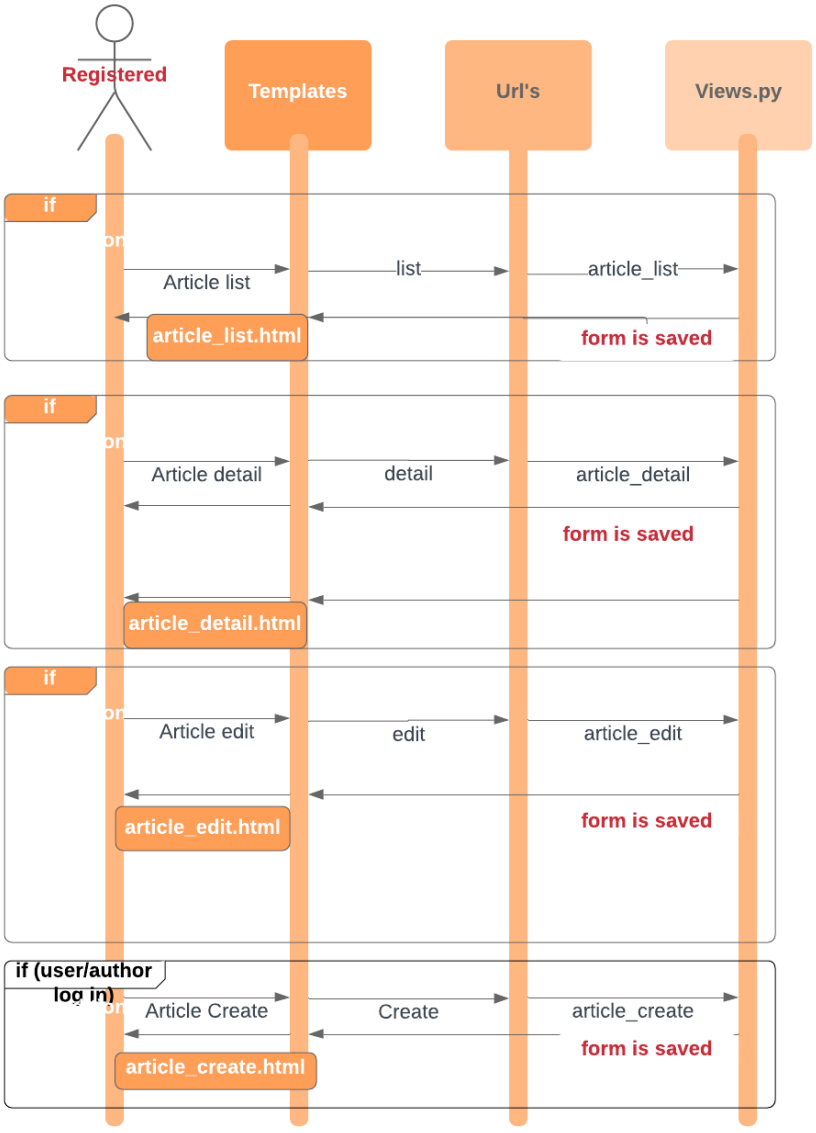

Figure 4 Article execution

5.5 Comment Section:

On this website, author can add the articles and visitor can comment on this site. Django includes a simple, yet customizable comments framework. The built-in comments framework can be used to attach comments to any model, so you can use it for comments on blog entries, photos, book chapters, or anything else.

To work on comment section, we start by following steps:

Models.py:

Model comments will contain the following fields:

Path- It will contain an array of integers, which will contain the full path to the root. As mentioned in the article on Materialized Path, is the ID of the parent element.

Post- a foreign key to an article in which there is a comment.

Author- a foreign key to the author's comment.

Text- message.

Created Date - date and time of the publication of the comment.

overridden method **__str__** , which will be responsible for the display of the contents of a comment in the admin area.

Create tables for models in your database:

Now it's time to add our comment model to the database. To do this we have to tell Django that we made changes to our model. Type
python manage.py makemigrations blog in your command line.
Our Comment model exists in the database now

Register Comment model in admin panel:

admin.site.register(Comment)

and don’t forget to import the comment model at top of the file

from .models**import** Article, Comment

Comment form:

The comment form will be placed in a separate file **forms.py**.

from django**import** forms

comment form will be created, and it will be connected to respective articles.

Make our comments visible:

we went to articles/templates/articles.aticle_detail.html and add function in html file to make it visible. We made class “comments” and call it by method “post” then gave {% csrf_token %} {{ comment_form.as_p }}
after this we implemented for loop to comment by user with date and text and then close it in {% endblock %}

But it could look a little bit better, so let's add some CSS to the bottom of the static/css/blog.css file:

```
.comment {
margin: 20px0px20px20px;
}
```

Chapter 6

Testing

Having tests for any project will helps you to find bugs. If any of the function breaks, you will now about it. Its easier to debug code line by line.

Unit Tests: Unit Test are isolated tests that test one specific function.
Test Case: A test case is executing set of features for your Application. Proper development of test cases finds problems in your functionality of an Application.
Test suite: A test suite is a collection of test cases. It is used to aggregate tests that should be executed together.

In general, tests result in either a Success (expected results), Failure (unexpected results), or an error. While writing test cases, not only testing for the expected results but also need to test how good your code handles for unexpected results.

what is anautomated test it's a piece of codewhich makes sure that the another pieceof code is working basically we writecode like we would normally do this codeisn't really doing anything in ourprogram itself it's just asserting thatthe other pieces of code are working andthere are different kinds of automatedtests but they all are very importantthat they run under certain condition sowe can check whether they are workingcorrectly under certain condition whyshould you test first of all it leads tohigher application quality and less bugsideally because we're making sure that apiece of code is working as I saidalready and that happens under certainconditions and we can always then rerunthe tests and make sure that they stillwork the next point is refactoringwhenever we are refactoring we areessentially changing the code withoutchanging the behavior that's thedefinition of refactoring and in orderto assert that the behavior is still thesame we need a way to of course assertthat testing and manually wouldn't makesense it would be way too time-consumingit would make sense but not alone andtherefore we have automated tests whichwe can run and then assert thateverything is still working as we expectit to be even after we refactor then italso allows for easier version upgradesbecause from version to version ofcourse things will change and if youhave automated tests you can just runthem and see what changed and then fixthe errorrun them again see that everything isworking fine and then you are good to golet's dive into how we approach testingwith an analogy in this picture we havea smoke detector so what can we do totest that this is working correctly thefirst condition would be that it doesn'tgo off if there's no smoke right that'sthis condition so under the conditionthat there is no smoke then thisshouldn't fire because otherwise we will get false alarms we will get intotrouble when there isn't really anythingso basically it needs to just be silentwhenever there is nothing to worry aboutin this case no smokebut under the condition that there issmoke we better make sure that thatthing actually fires and wakes up everybody in the house,so again under the condition that thereis smoke we make sure that this thinggoes off that would be the next testright this is of course very simplifiedwe would have different gradients

ofcourse to when it goes off and when notwhich would also need to be tested thisis again just simplified just toillustrate how we go generally abouttesting things let's take a look at the different types of tests the first oneour unit tests they test one piece independently of any other pieces let'swhat they're good for and they are thefirst to run by the way also one example for unit tests would be this so we havea function which is at numbers it takesan eighth first and second parameter andthen returns first plus second then wehave a function called test add numbers we usually circular to assert that theresult of add numbers what it returns ifwe pass in five and three will be eightthe second word is pretty simple if weassert something that is true then it doesn't complain it doesn't do anythingbut if we assert something false in thiscase we assert that one equals two thenwe get an assertion error the secondtypes of tests are integration teststhey test multiple pieces together toassert that they work correctly with oneanother a simple example for that wouldbe the request response cycle in Djangofor example why we can use the Django testclient we're going to do that throughoutthe series by the way so in the case ofrequest response examples of course manymore than one component is involved thenwe have functional tests which aresometimes calledacceptance tests they test saideverything works from the end userspoint of view and they therefore othersclose run because we in the case of webapplications we basically mimic thebrowser behavior when a user would typeinto a form so actually when doingfunction tests.

Testing the Forms:

setUp(): The setUp() methods allows to define instructions which will be executed before and after each test method.
Every test case method should start with "test", because When you run your tests, the default behavior of the test utility is to find all the test cases in all your files, function name starts with test, which automatically build a test suite out for all test cases and run.

For Testing any Form, In "Setup_Class" we create required objects here, and will test whether the created object details matched or not.assertTrue() or assertFalse() functions is to verify condition which returns "True/False"

In the above functions in class "User_Form_Test" returns True/False based on the input data given.

6.1 Testing the Views:

When we start testing the vews, first test for the response codes then we got with the actual response.

assertEqual(): assertEqual() to check for an expected result.
assertTemplateUsed(): This Assert is used for rendering the response.

First, how do I know exactly what to test? Is there a tool for highlighting un-tested code? Yes, and that tool is called “test coverage” which is also the general idea behind a lot of powerful tools in the same category.

In Python we can install a tool called coveragepip install coverage

And run coverage check with:

```
1. coverage run --source='.' manage.py test
```

Conclusion:

This website designed to provide information about fitness, health and exercise tips as it is one of most important aspect for every human being, so we put our efforts to make it best.

After working on this amazing project, we learnt a lot and we got better understanding about python with Django framework.

We came to know how to design a website and how it is working as we are working on different parts like frontend with html, CSS, bootstrap.

Then we worked on backend with python programming and created a website.

So now we are quite confident that we can create any type of website using python.

References:

1. https://docs.djangoproject.com/en/2.2/
2. https://django-taggit.readthedocs.io/en/latest/
3. https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-django-project.html
4. https://www.tutorialspoint.com/python/
5. https://micropyramid.com/blog/django-unit-test-cases-with-forms-and-views/
6. https://docs.djangoproject.com/en/2.2/howto/static-files/
7. https://www.valentinog.com/blog/testing-django/
8. https://micropyramid.com/blog/django-unit-test-cases-with-forms-and-views/
9. https://docs.djangoproject.com/en/2.2/howto/static-files/