INTERNSHIP REPORT

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

by

CHANDAN S

Under Supervision of

Sumit Jha

Assistant manager

Landmark Group, Bengaluru.

ACKNOWLEDGEMENT

First, I would like to thank **Navraj Arora**, the HR Manager of **Landmark Group** for giving me the opportunity to do an internship within the organization.

I also would like all the people that worked along with me **Landmark Group** with their patience and openness they created an enjoyable working environment.

It is indeed with a great sense of pleasure and immense sense of gratitude that I acknowledge the help of these individuals.

I am highly indebted to **Sumit Jha** (Assistant manager) for the facilities provided to accomplish this internship.

I would like to thank my Manager **Harshith Raj** for his support throughout my internship.

ABSTRACT

The Company:

Landmark Group, founded in 1973, has a diverse portfolio of retail and hospitality brands. It has successfully grown into one of the largest and most successful retail conglomerates in the MENA region with an entrepreneurial culture focused on delivering exceptional value. The Group employs over 45,000 people, and operates over 1,600 outlets covering 22 million square feet across 19 countries in the Middle East, North Africa and India region. Its vast portfolio of successful businesses includes award-winning household brands like Lifestyle, Max, Splash and Home Centre.

Landmark Group Data Labs was established in 2015 and is a key function to aid the different businesses of Landmark Group in both overall strategy and intelligent data driven decision making.

Methodology:

This project is to provide classifieds information. The user should register to utilize the site. Each user will be given User Id and password. Using that Id and password user can enter in to the site. This project is implemented using Html, CSS, JavaScript as the front-end and Django, sqlite3 and MySQL as back-end.

Learning Objectives/Internship Objectives

- Internships are generally thought of to be reserved for college students looking to gain experience in a particular field. However, a wide array of people can benefit from Training Internships in order to receive real world experience and develop their skills.
- An objective for this position should emphasize the skills you already possess in the area and your interest in learning more
- Internships are utilized in a number of different career fields, including architecture, engineering, healthcare, economics, advertising and many more.
- Some internship is used to allow individuals to perform scientific research while others are specifically designed to allow people to gain first-hand experience working.
- Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs. When you are applying for a Training Internship, make sure to highlight any special skills or talents that can make you stand apart from the rest of the applicants so that you have an improved chance of landing the position.

1.INTRODUCTION

In this website user will register by using the register number and pin provided to him/her. Initially the registered user will not be active user. Once the user gets registered a request will be sent for a validation to the admin. Admin can either validate or decline the user request.

1.1 Functionalities:

user will get an alert page based on the status of the registration.

user will get an email post admin action on the validation of the account.

Using multiple databases (sqlite3 & MySQL)

1.2 MODULES:

1. Home page
2. Admin Login page
3. Registration page
4. Login form
5. Sign up form
6. Validation tab

2. SYSTEM REQUIREMENTS SPECIFICATIONS

2.1 System configurations

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behavior, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

2.2 Software requirements:

Operating System: Windows

Coding Language: HTML, CSS, Django, JavaScript, and Bootstrap.

Text Editor : Visual studio code.

Database : My SQL, Sqlite3.

2.3 Hardware Requirements:

Processor : Intel core i3

Memory : 8GB RAM

Hard Disk : 1TB

3. TECHNOLOGY

3.1 Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source. Python-based free and open-source web framework that follows the model–template–views (MTV). Django was designed to help developers take applications from concept to completion as quickly as possible. Django takes security seriously and helps developers avoid many common security mistakes. Some of the busiest sites on the web leverage Django’s ability to quickly and flexibly scale.

Django's configuration system allows third party code to be plugged into a regular project, provided that it follows the reusable app conventions. More than 2500 packages are available to extend the framework's original behavior, providing solutions to issues the original tool didn't tackle: registration, search, API provision and consumption, CMS, etc. This extensibility is, however, mitigated by internal components' dependencies. While the Django philosophy implies loose coupling, the template filters and tags assume one engine implementation, and both the auth and admin bundled applications require the use of the internal ORM. None of these filters or bundled apps are mandatory to run a Django project, but reusable apps tend to depend on them, encouraging developers to keep using the official stack in order to benefit fully from the app’s ecosystem.

3.2 HTML

HTML is the standard mark-up language for creating Web pages.

HTML stands for Hyper Text Mark-up Language

HTML describes the structure of Web pages using mark-up

HTML elements are the building blocks of HTML pages

HTML elements are represented by tags

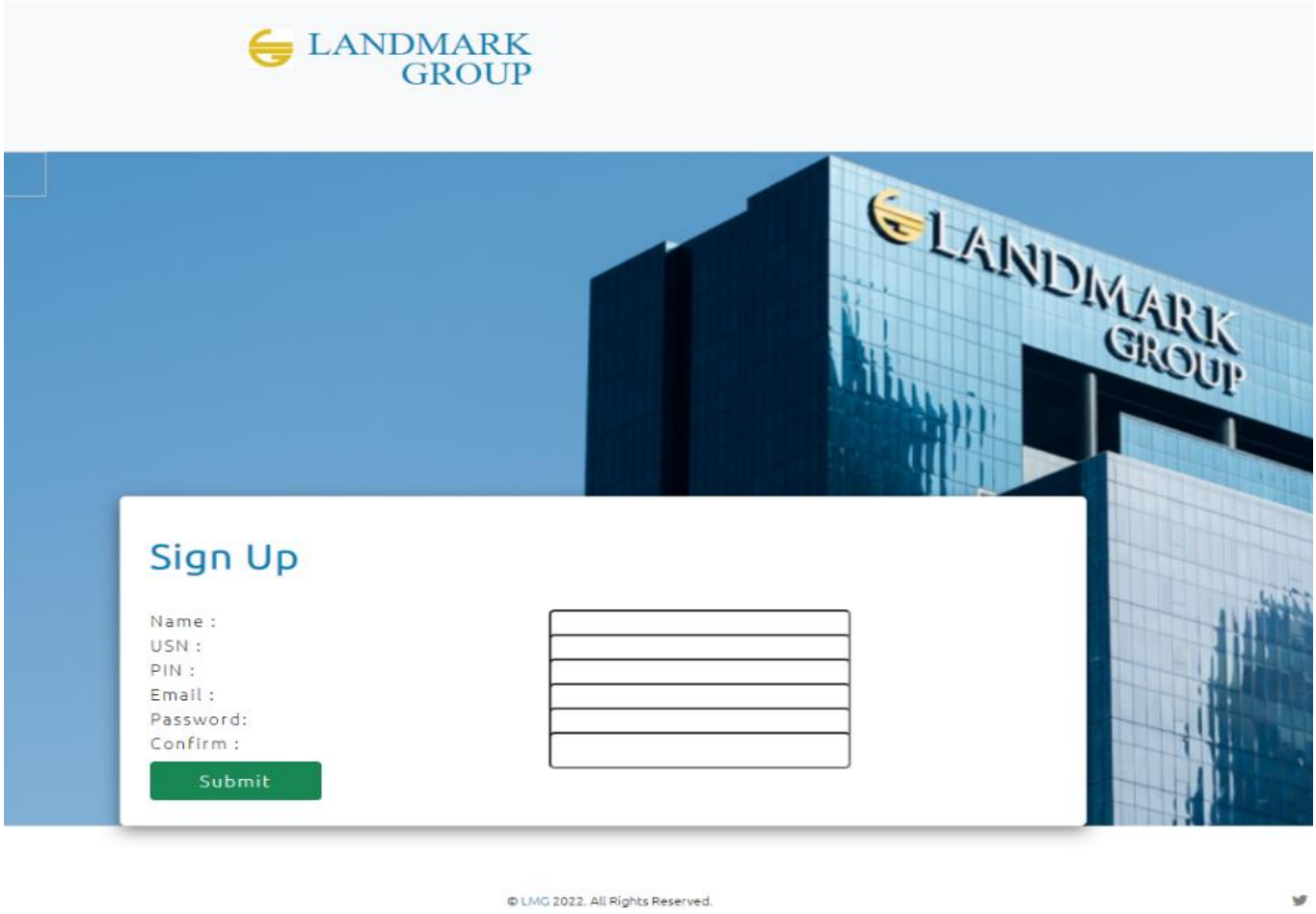
HTML tags label pieces of content such as "heading", "paragraph", "table", and so on

CHAPTER 1

MODELS AND VIEWS

Sign up:

The sign-up page consists of name, usn, pin, email and password fields, the inputs for this fields is taken through html forms.



forms

```
{% extends 'base.html' %}
        {% block content %}
            <h3><span style="color:#0983b7; font-family: 'Ubuntu', sans-serif; font-
size: larger; font-weight: bolder;margin-bottom: 5%;">
                Sign Up</span></h3><br>
            <!-- <p>Login in to see it in action.</p> -->
            <form  id="f1"  method="POST">
                {% csrf_token %}
                <label for="">Name   :</label>
                <input type="text" name="name" id="id_username" required><br>
                <label for="">USN   :</label>
                <input type="text" name="usn" id="id_username" required><br>
                <label for="">PIN   :</label>
                <input type="text" name="pin" id="id_username" required><br>
                <label for="">Email   :</label>
                <input type="email" name="email" class="id_password" required><br>
```

```
                <label for="">Password:</label>
                <input type="password" name="pwd" class="id_password" id="pwd "
required><br>
                <label for="">Confirm :</label>
                <input type="password" name="cpwd" class="id_password" id="cpwd"
required><br>
                <div id="error"> </div>
                <span style="display:block; height: 5px;"></span>
                <input style="letter-spacing:1.5px; font-family: 'Ubuntu', sans-serif;
padding-left: 40px; padding-right: 40px;" id="sub" type="submit" class="btn btn-
success">
            </form>
        {% endblock content %}
```

Data will be stored to database by creating a model in models.py in Django user application. Where every row or single user data will be stored as an object of the class user. Here the user initially will not be active user to make sure that, Check attribute is made default to False. Check will become True once after admin validate it. Below fig show check is False by red cross mark

☐ sumith	1EC045	123456	SUMIT@GMAIL.COM	123	✗

models.py

```
from django.db import models

class user(models.Model):
    name=models.CharField(null=True,blank=True,max_length=100)
    usn=models.CharField(max_length=100,primary_key=True,null=False)
    pin=models.IntegerField(blank=False,null=False)
    email=models.EmailField(max_length=100)
    password=models.CharField(max_length=20)
    Check=models.BooleanField(default=False)
```

After the data is entered into forms, the POST request will be sent into view function named inside views.py in Django user application and user object will be saved into database.

views.py

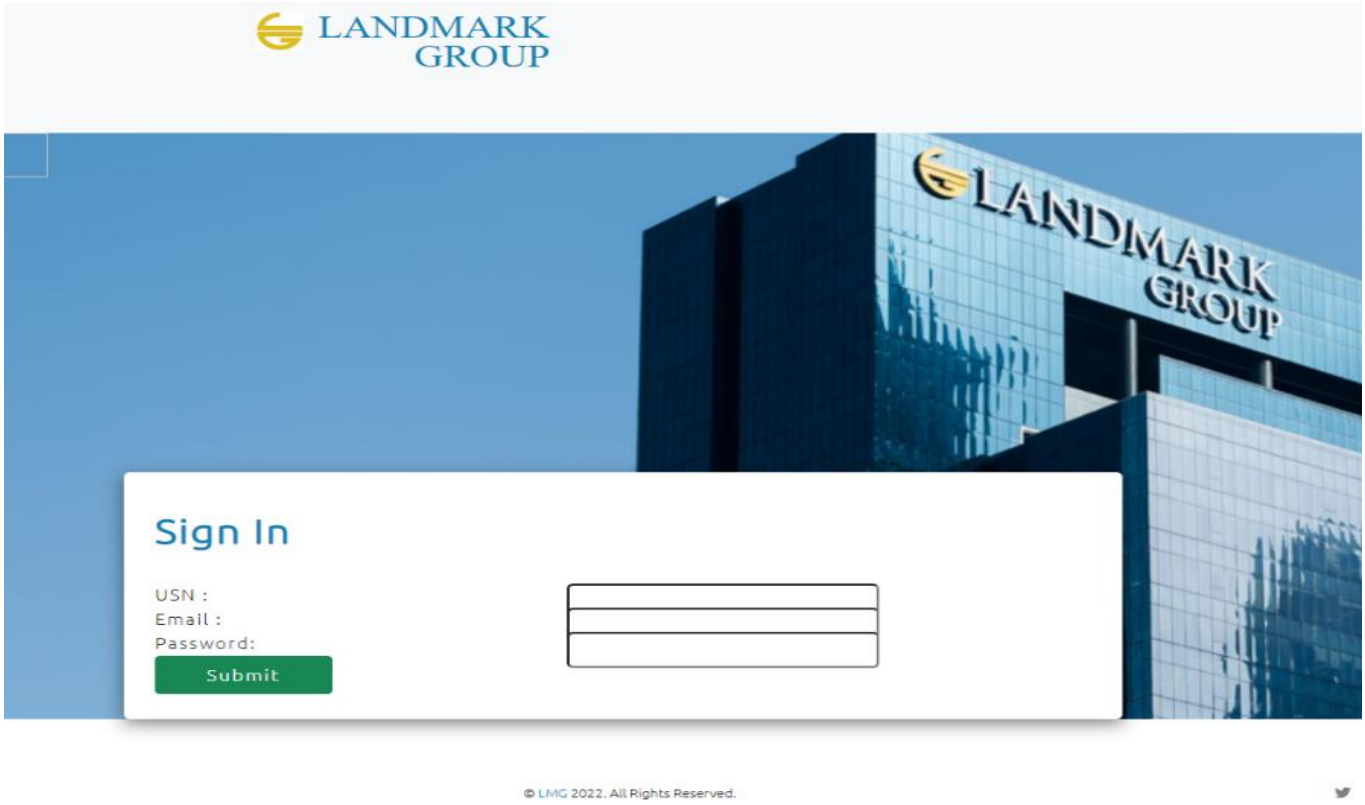
```
def insert(request):
    ob=user()
    na=request.POST['name']
    ob.name=request.POST['name']
    ob.usn=request.POST['usn']
    ob.pin=request.POST['pin']
    ob.email=request.POST['email']
    ob.password=request.POST['pwd']
    ob.save()
    return render(request,'next.html',{'name':na})
```

CHAPTER 2

URLS

Sign in:

The sign-in page consists of name, usn, email and password fields, the inputs for this fields is taken through html forms




```
{% extends 'base.html' %}
{% block content %}
<h3><span style="color:#0983b7; font-family: 'Ubuntu', sans-serif; font-size: larger;
font-weight: bolder;margin-bottom: 5%;">
    Sign In</span></h3><br>
    <form  id="f1" action="check" method="post">
        {% csrf_token %}
        <label for="">USN   :</label>
        <input type="text" name="usn" id="id_username"><br>
        <label for="">Email   :</label>
        <input type="email" name="email" class="id_password"><br>
        <label for="">Password:</label>
        <input type="password" name="pwd" class="id_password" ><br>
        <input style="letter-spacing:1.5px; font-family: 'Ubuntu', sans-serif; padding-
left: 40px; padding-right: 40px;" type="submit" class="btn btn-success">
    </form>
{% endblock content %}
```

Data entered in the forms will be retrieved through POST request which will be then searched in the database whether the user data is present in the database, if present it will redirect it next page.

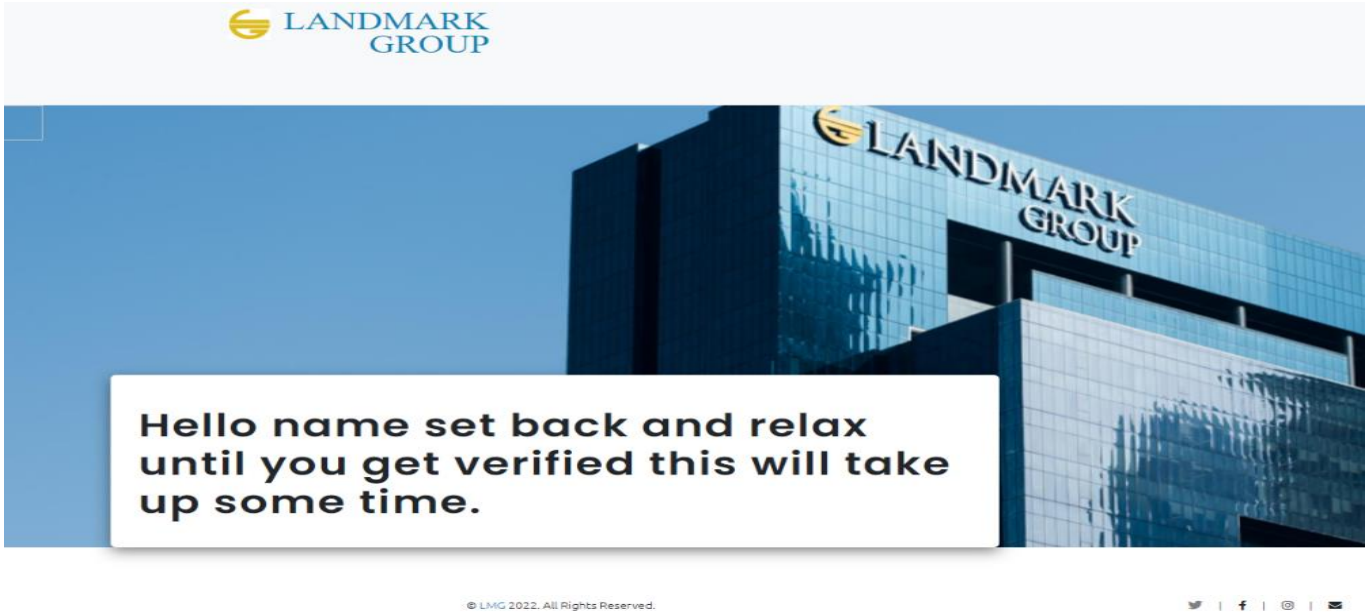
views.py

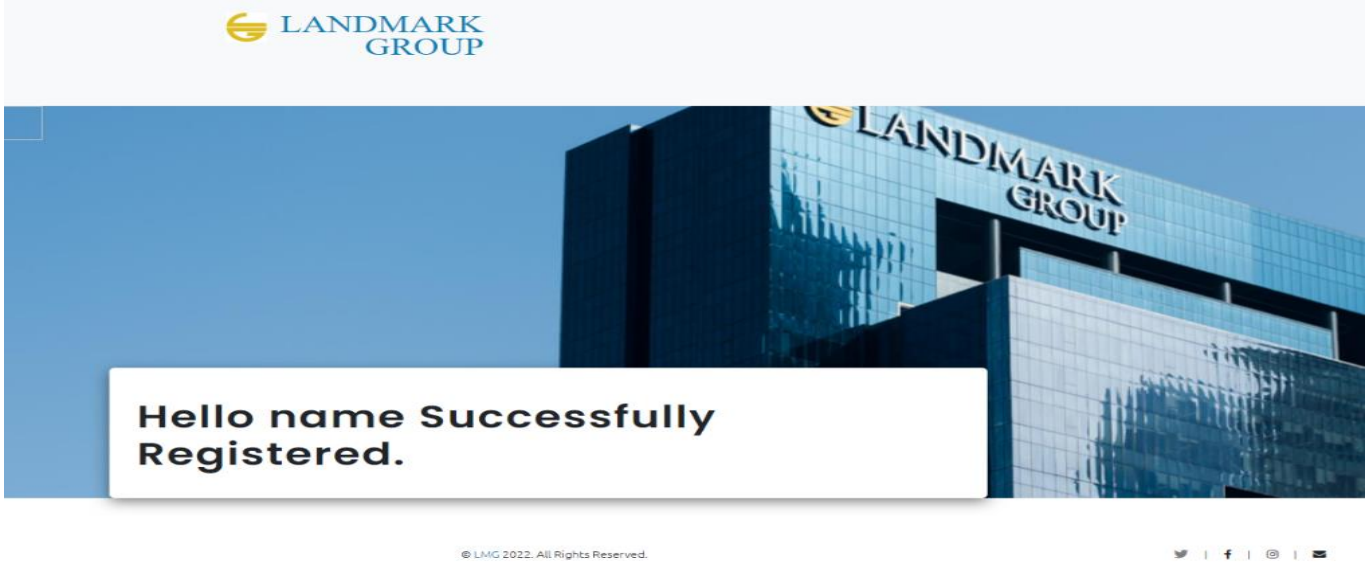
```
def check(request):
    pi=user.objects.get(pk=request.POST['usn'])
    if pi!=NULL and pi.password==request.POST['pwd']:#checking if the object exist
         return render(request,'next.html',{'data':pi})
    else:
         return render(request,'nouser.html')
```

next:

The next page will display the alert based on the user candidature status with respective message.

```
{% extends 'base.html' %}
{%block content %}
<h1>Oops {{data}} no user found check id or password</h1>
{% endblock %}
```






```
{% extends 'base.html' %}


{%block content %}

{% if data %}
    {% if data.check %}
    <h1>Hello {{data.name}} Successfully Registered.</h1>
    {% else %}
    <h1>Hello {{data.name}} set back and relax until you get verified this will take up
some time.</h1>
    {% endif %}
{% else %}
<h1>Hello {{name}} set back and relax until you get verified this will take up some
time.</h1>
{% endif %}
{% endblock %}
```

urls.py

For every action in the form or href in anchor tags or whichever leads to path or urls to redirect it will be sequentially checked in urlpatterns list with the first value if match is found it maps to the function that matches in the views.py. here action check will be mapped to check function in the view which is present in 8th position

```
from django.contrib import admin
from django.urls import path
from user import views
from archive import views as vw

# Function views
#     1. Add an import:  from my_app import views
```

```
#     2. Add a URL to urlpatterns:  path('', views.home, name='home')
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.home,name="show"),
    path('index',views.home,name="show"),
    path('signup',views.signup),
    path('signin',views.signin),
    path('asignin',views.asignin),
    path('insert',views.insert),
    path('check',views.check),
    path('check1',views.check1),
    path('update',views.update_data,name="updatedata"),
    path('delete',views.delete,name="deletedata"),
    path('Ainsert',vw.Ainsert),
    path('archive',vw.signup),]
```

CHAPTER 3

DJANGO EMAILS

ADMIN

Admin sign-in:

The sign-in page consists of email and password fields, the inputs for this fields is taken through html forms

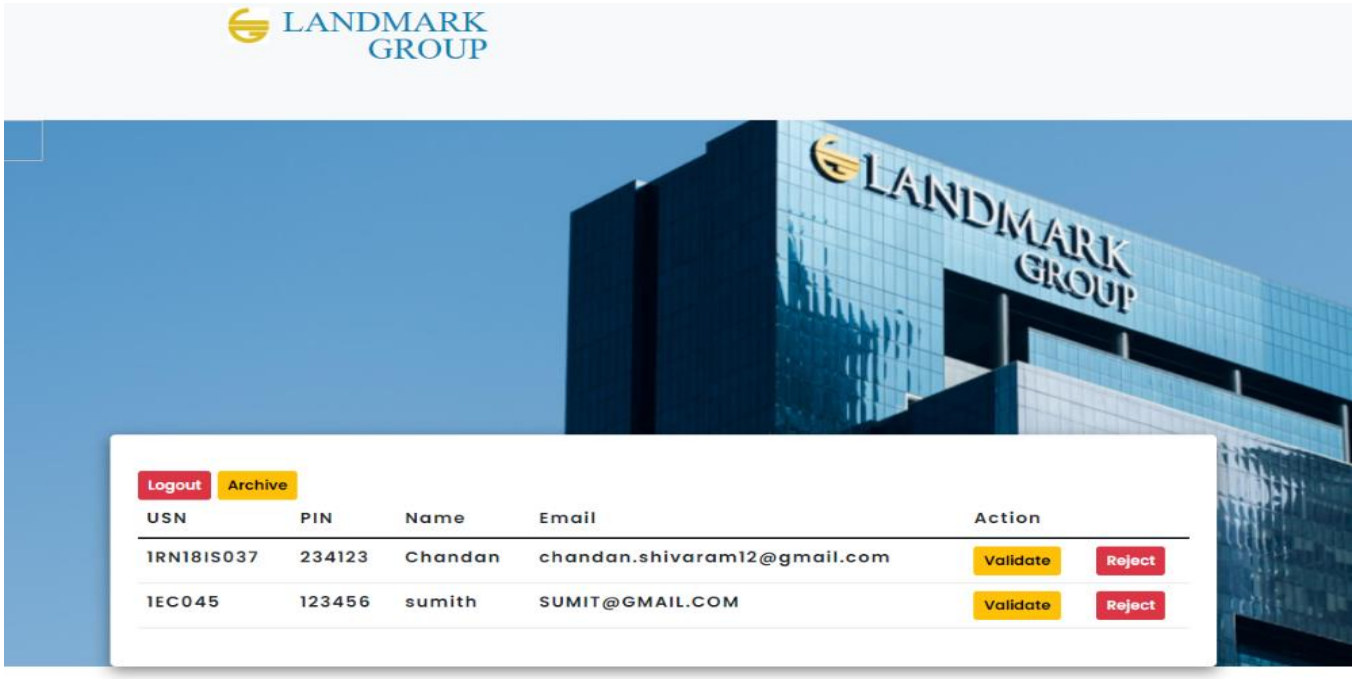
Data entered in the forms will be retrieved through POST request which will be then searched in the database whether the user data is present in the database, if present it will redirect it next page

views.py

```
def check1(request):
    ob=user.objects.all()
    ad=User.objects.get()
    if ad.email==request.POST['email'] and
check_password(request.POST['pwd'],ad.password):
        return render(request,'admin.html',{'data':ob})
    else:
        return render(request,'nouser.html')
```

ADMIN PANEL:

This panel will contain all the newly registered users (invalid user) data in a table where each row represents single user. Here data is retrieved by iterating object which contains all the user's object.



```
ob=user.objects.all()
```

```
{% extends 'base.html' %}


{%block content %}
<form action="index" method="post" class="d-inline">
  {% csrf_token %}
    <input  type="submit"  value="Logout" class="btn btn-danger btn-sm">
</form>
<form action="archive" method="post" class="d-inline">
  {% csrf_token %}
    <input  type="submit"  value="Archive" class="btn btn-warning btn-sm">
</form>
{% if data %}
        <table class="table">
            <thead>
              <tr>
                <th scope="col">USN</th>
                <th scope="col">PIN</th>
                <th scope="col">Name</th>
                <th scope="col">Email</th>
                <th scope="col">Action</th>
              </tr>
            </thead>
            <tbody>
                {% for st in data %}
                {% if st.check is False %}
                <tr>
                    <th scope="row">{{st.usn}}</th>
                    <td>{{st.pin}}</td>
                    <td>{{st.name}}</td>
                    <td>{{st.email}}</td>
                    <td>
                        <form action="update" method="post" class="d-inline">
                            {% csrf_token %}
                            <input type="hidden" name="id" value="{{st.usn}}">
                            <input  type="submit" value="Validate" class="btn btn-
warning btn-sm">
                        </form>
                    </td>
                    <td>
                      <form action="delete" method="post" class="d-inline">
                          {% csrf_token %}
                          <input type="hidden" name="id" value="{{st.usn}}">
                          <input  type="submit" value="Reject" class="btn btn-danger
btn-sm">
                      </form>
                  </td>
                {% endif %}
                </tr>
```

```
            {% endfor %}
          </tbody>
        </table>
{% else %}
        <h4 class="text-center">No Records to Validate</h4>
        {% endif %}
        <script>
          $(document).ready(
            function(){
              $(".nav-link").text("");
              $(".middle-box").css("width","1000px");
            }
          );
        </script>
{% endblock %}
```

Validate

Here admin can either validate or reject the user. If admin validate the user the user check attribute will be changed to **True** and the user will become active user. The user object will be retrieved using **id** which is **primary key** and check attribute will be changed to **True**.

USN	PIN	Name	Email	Action	
1RN18IS037	234123	Chandan	chandan.shivaram12@gmail.com	Validate	Reject
1EC045	123456	sumith	SUMIT@GMAIL.COM	Validate	Reject

views.py

```
def update_data(request):
        ob=user.objects.all()
        pi=user.objects.get(pk=request.POST['id'])
        send_mail(
            'Congrats account verified',
            'Hi '+pi.name+' your account is verified \n Now you are active user.',
            'chandan.s@landmarkgroup.in',
            [pi.email],
        )
        pi.check=True
        pi.save()
        return render(request,'admin.html',{'data':ob})
```

Email:

Here user will get an email after admin validate or reject that user. Mail is sent using the SMTP host and port specified in the **EMAIL_HOST** and **EMAIL_PORT** settings.
The **EMAIL_HOST_USER** and **EMAIL_HOST_PASSWORD** settings, if set, are used to authenticate to the SMTP server, and the **EMAIL_USE_TLS** and **EMAIL_USE_SSL** settings control whether a secure connection is used in **settings.py**.

settings.py

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = email
EMAIL_HOST_PASSWORD = 'ugnvgkwdfauzemez'
```

In views.py the mail is sent using send_mail module which will be present in django.core.mail. Here the email will be sent to user after admin validate or reject the user.

views.py

```
from django.core.mail import send_mail

#send_mail(subject, message, from_ email, recipient_email)

def update_data(request):
        ob=user.objects.all()
        pi=user.objects.get(pk=request.POST['id'])
        send_mail(
            'Congrats❄ account verified',
            'Hi '+pi.name+' your account is verified \n Now you are active user.',
            'chandan.s@landmarkgroup.in',
            [pi.email],
        )

def delete(request):
        ob=user.objects.all()
        pi=user.objects.get(pk=request.POST['id'])
        send_mail(
            'Sorry 😔 account rejected',
            'Hi '+pi.name+' your account got rejected \nPlease check the information and
register again.',
            'chandan.s@landmarkgroup.in',
            [pi.email],
        )
        pi.delete()
        return render(request,'admin.html',{'data':ob})
```

Sorry 😌 account rejected Bin ×

1rn18cs034.chandanshivaram@gmail.com
to me

Hi chandan s your account got rejected
Please check the information and register again.

Congrats🎇 account verified Bin ×

1rn18cs034.chandanshivaram@gmail.com
to me

Hi chandan s your account is verified
Now you are active user.

CHAPTER 4

SIGNALS

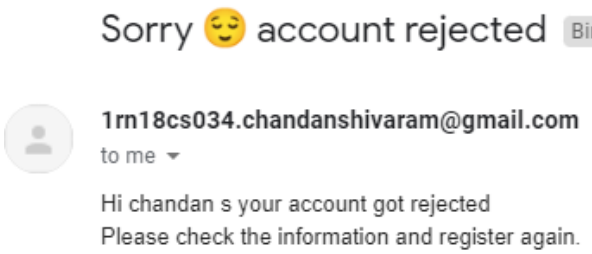
Admin reject:

Here signals is implemented for admin reject action on user request. After admin reject the user request the data will be deleted in user table in the database and it will also save in deleted_user table pre_delete.

The delete method inside the views.py will get the object of the user whose data is to be deleted by its id.

views.py

```
def delete(request):
        ob=user.objects.all()
        pi=user.objects.get(pk=request.POST['id'])
        send_mail(
            'Sorry 😌 account rejected',
            'Hi '+pi.name+' your account got rejected \nPlease check the information and
register again.',
            'chandan.s@landmarkgroup.in',
            [pi.email],
        )
        pi.delete()
        return render(request,'admin.html',{'data':ob})
```



Signals:

To insert the deleted data into deleted_user table, the model has to be defined in models.py

models.py

```
class deleted(models.Model):
    name=models.CharField(null=True,blank=True,max_length=100)
    usn=models.CharField(max_length=8,primary_key=True,null=False)
    pin=models.IntegerField(blank=False,null=False)
    email=models.EmailField(max_length=100)
    date = models.DateTimeField(null=True, blank=True)
```

After admin select reject button the **.delete()** method will trigger signals. Before the data get deleted pre_delete signal will be triggered this will be implemented in signals.py file which is inside the signals. Before that it has to be defined in apps.py.

apps.py

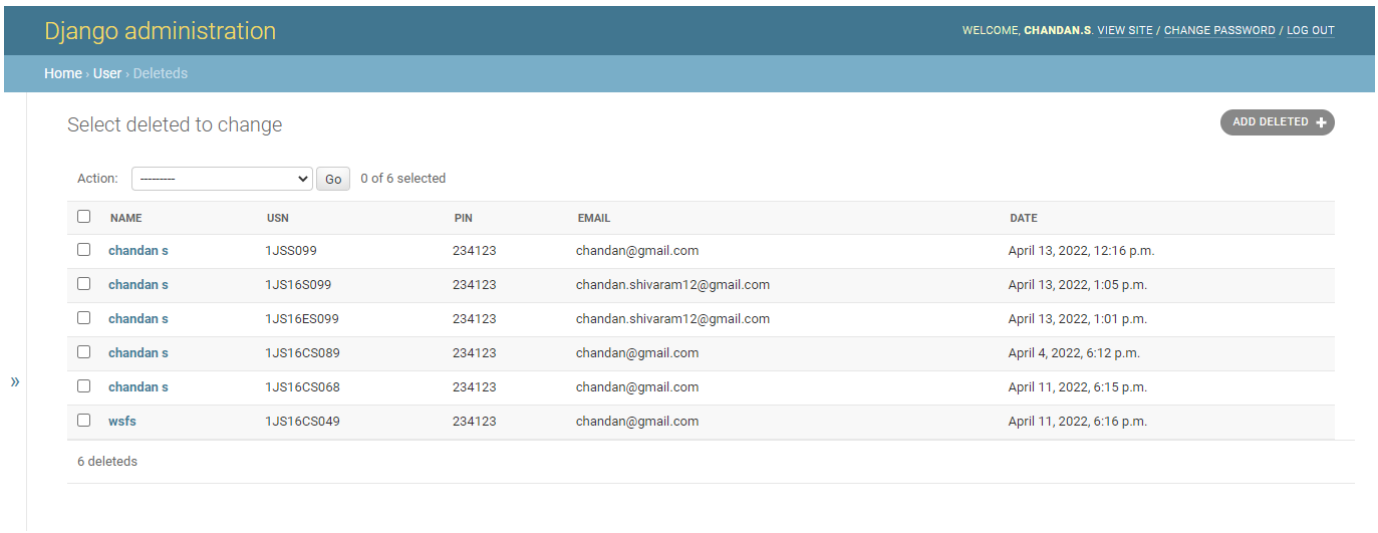
```
def ready(self):
        import user.signals
```

After pre_delete triggered control goes into create_table. Here the object of the class deleted will be created and data of the object will be assigned by retrieving the data from the instance which will deleted from user table, which will be passed into create_table by pre_delete method

signals.py

```
from django.db.models.signals import pre_delete
from .models import deleted
from django.dispatch import receiver
import datetime

@receiver(pre_delete,sender=user)
def create_table(sender, instance, **kwargs):


        d=deleted()
        d.name=instance.name
        d.usn=instance.usn
        d.email=instance.email
        d.pin=instance.pin
        d.date=datetime.datetime.now()
        d.save()
        print('Object inserted to deleted table')
```

The below figure shows the data get deleted will be inserted into deleted table.



CHAPTER 5

DATABASE ROUTING

Database routing:

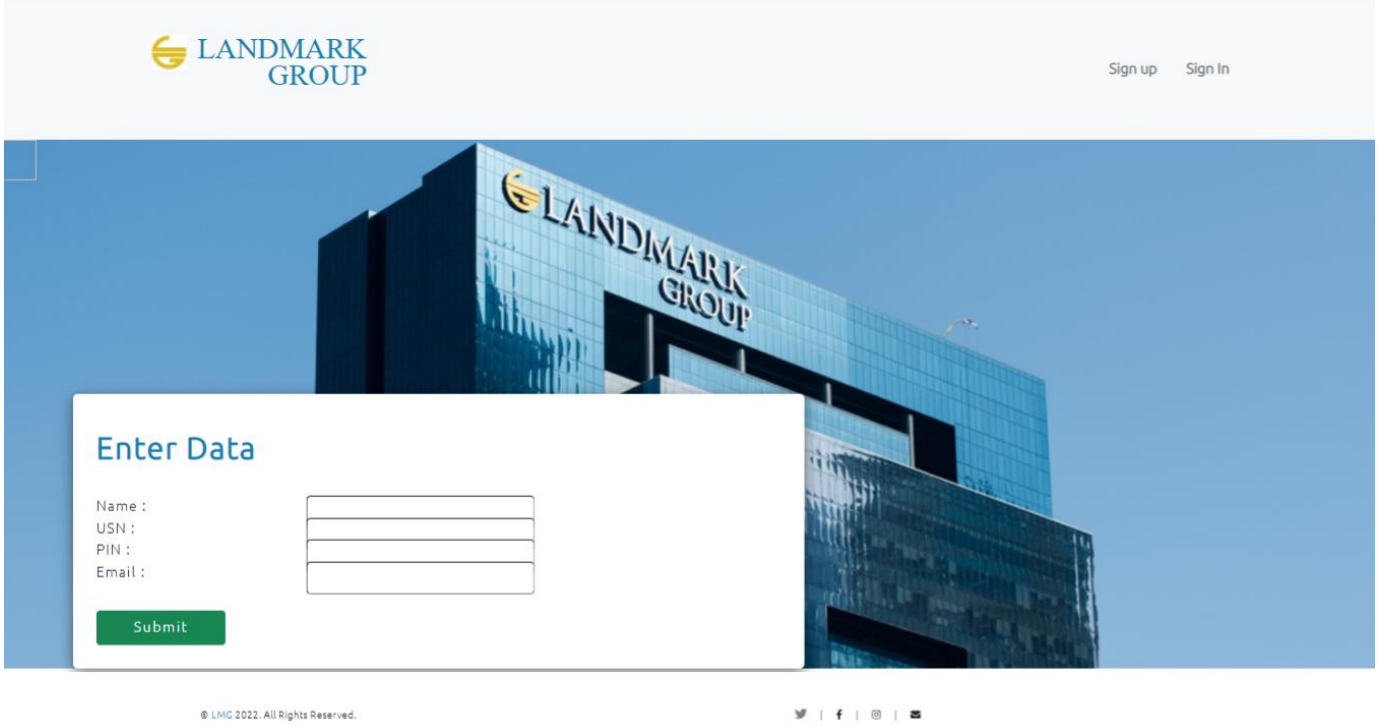
Till now I have used default sqlite3 database for faster access of data. Here admin can also insert a data into MySQL database to archive the data. This is done through creating another app “archive”. In archive a model is created to create a archive table in MySQL database.

Admin can archive data after login into the admin panel and access it by clicking on archive button.

archive/models.py

```
from django.db import models

# Create your models here.
class user(models.Model):
    name=models.CharField(null=True,blank=True,max_length=100)
    usn=models.CharField(max_length=100,primary_key=True,null=False)
    pin=models.IntegerField(blank=False,null=False)
    email=models.EmailField(max_length=100)
```

After admin click on archive button, the below form will be displayed to enter the data to be archived. Here data is inserted similar to user app.



archive/views.py

```
from django.shortcuts import render
from archive.models import user
# Create your views here.
def Ainsert(request):
    ob=user()
    na=request.POST['name']
    ob.name=request.POST['name']
    ob.usn=request.POST['usn']
    ob.pin=request.POST['pin']
    ob.email=request.POST['email']
    ob.save()
    return render(request,'Anext.html',{'name':na})
```

Here the facility to choose between the database is implemented in db_routers.py which is inside the routers directory which is in the project folder. Before that it is to defined in settings.py which done by initializing the router class name which correspond to the application in a list called **DATABASE_ROUTERS** where the value is class name in below list routers.db_routers.AuthRouter represent the implementation of user application database in which it uses.

```
DATABASE_ROUTERS = [
    'routers.db_routers.AuthRouter',
    'routers.db_routers.ArchiveRouter',
]
```

All required databases should be defined in settings.py via DATABASES dictionary where value is the databases used in applications. Here the database “default” is sqlite3 which used by user application for CRUD operation and “archive_db” is used by archive application to insert the archive data.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    },
    'archive_db':{
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'archive',
        'USER': 'root',
        'PASSWORD': password,
        'PORT':3306
    }
}
```

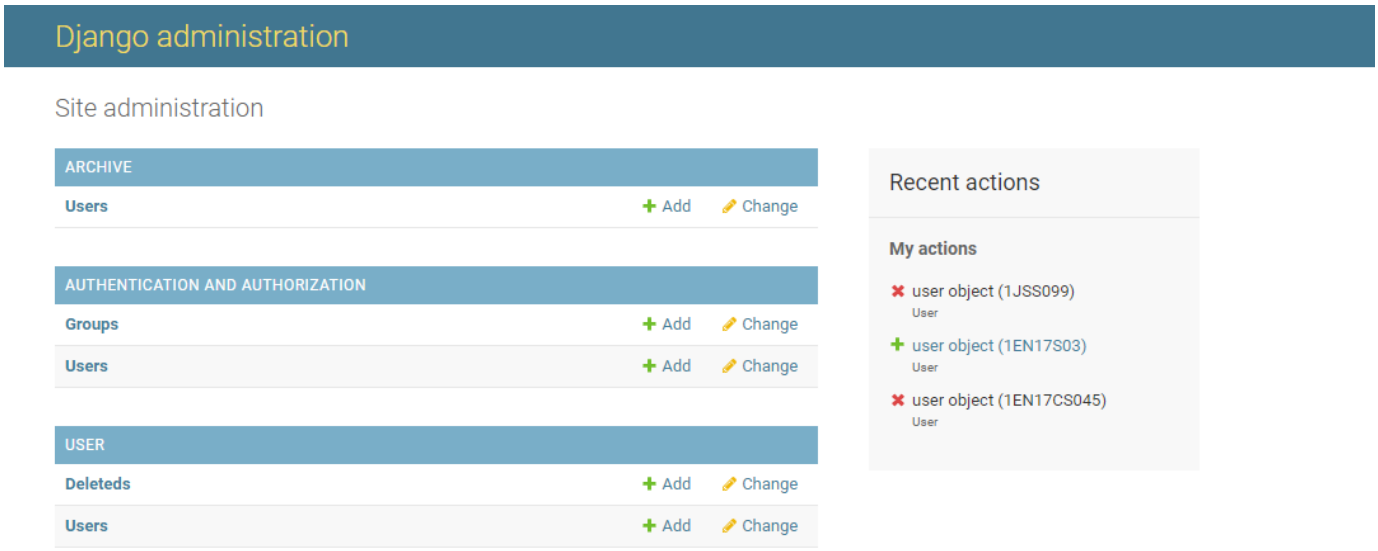
Here route_app_labels contain the apps name and if the apps exist it will return the mentioned database. Here AuthRouter returns “default” database for apps user, admin and session. And ArchiveRouter will return “archive_db” for archive app.

routers/db_routers.py

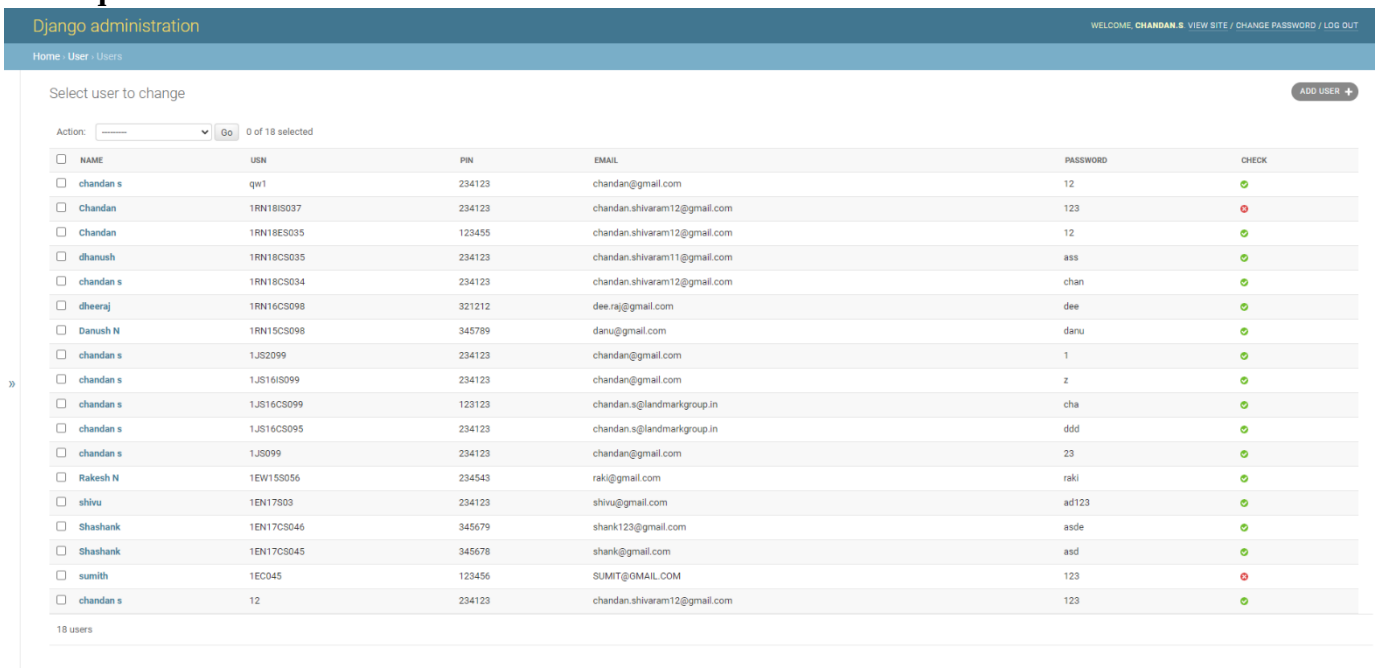
```
class AuthRouter:
    """
    A router to control all database operations on models in the
    user application.
    """
    route_app_labels = {'user', 'admin', 'sessions'}

    def db_for_read(self, model, **hints):
        """
        Attempts to read user  go to default.
        """
        if model._meta.app_label in self.route_app_labels:
            return 'default'
        return None

    def db_for_write(self, model, **hints):
        """
        Attempts to write user and contenttypes models go to default.
        """
        if model._meta.app_label in self.route_app_labels:
            return 'default'
        return None

    def allow_relation(self, obj1, obj2, **hints):
        """
        Allow relations if a model in the user app is
        involved.
        """
        if (
            obj1._meta.app_label in self.route_app_labels or
            obj2._meta.app_label in self.route_app_labels
        ):
            return True
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        """
        Make sure the auth and contenttypes apps only appear in the
        'default' database.
        """
        if app_label in self.route_app_labels:
            return db == 'default'
        return None

class ArchiveRouter:
    route_app_labels = {'archive'}

    def db_for_read(self, model, **hints):
        if model._meta.app_label in self.route_app_labels:
            return 'archive_db'
        return None
```

```python
    def db_for_write(self, model, **hints):
        if model._meta.app_label in self.route_app_labels:
            return 'archive_db'
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        if app_label in self.route_app_labels:
            return db == 'archive_db'
        return None
```

Databases:

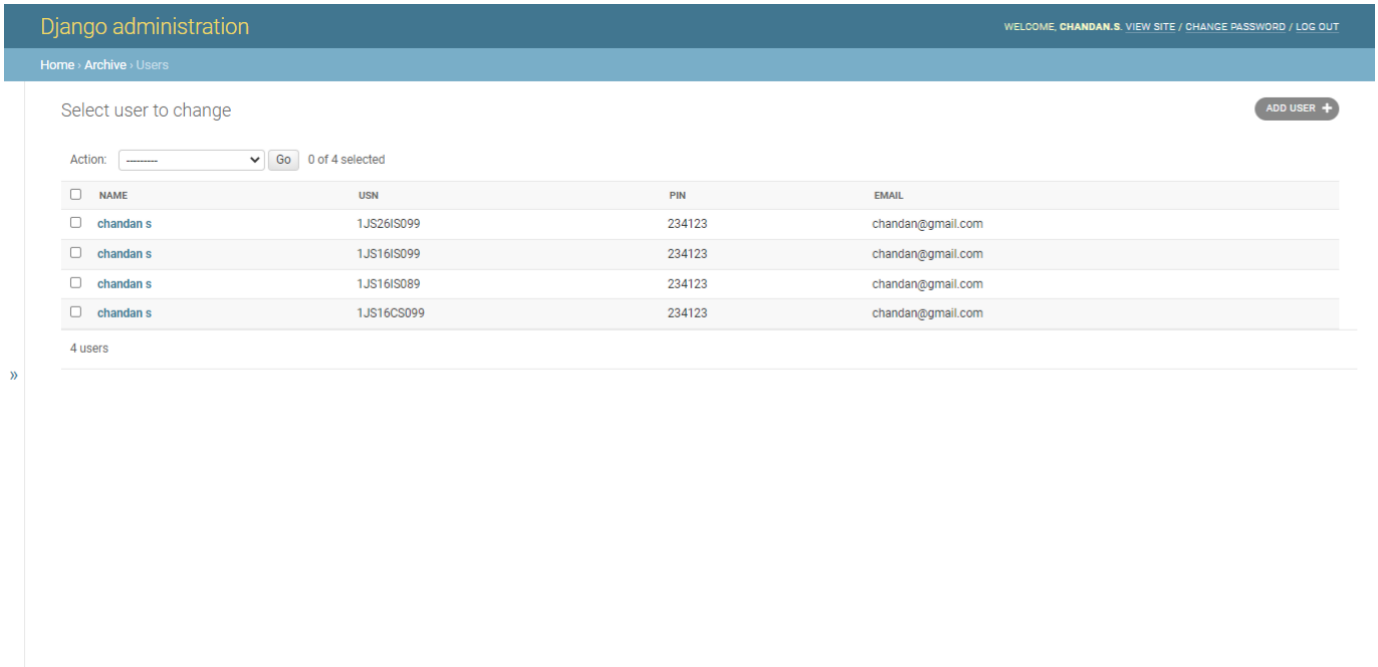


User sqlite3 database:



NAME	USN	PIN	EMAIL	PASSWORD	CHECK
chandan s	qw1	234123	chandan@gmail.com	12	✓
Chandan	1RN18IS037	234123	chandan.shivaram12@gmail.com	123	✗
Chandan	1RN18ES035	123455	chandan.shivaram12@gmail.com	12	✓
dhanush	1RN18CS035	234123	chandan.shivaram11@gmail.com	ass	✓
chandan s	1RN18CS034	234123	chandan.shivaram12@gmail.com	chan	✓
dheeraj	1RN16CS098	321212	dee.raj@gmail.com	dee	✓
Danush N	1RN15CS098	345789	danu@gmail.com	danu	✓
chandan s	1JS2099	234123	chandan@gmail.com	1	✓
chandan s	1JS16IS099	234123	chandan@gmail.com	z	✓
chandan s	1JS16CS099	123123	chandan.s@landmarkgroup.in	cha	✓
chandan s	1JS16CS095	234123	chandan.s@landmarkgroup.in	ddd	✓
chandan s	1JS099	234123	chandan@gmail.com	23	✓
Rakesh N	1EW15S056	234543	raki@gmail.com	raki	✓
shivu	1EN17S03	234123	shivu@gmail.com	ad123	✓
Shashank	1EN17CS046	345679	shank123@gmail.com	asde	✓
Shashank	1EN17CS045	345678	shank@gmail.com	asd	✓
sumith	1EC045	123456	SUMIT@GMAIL.COM	123	✗
chandan s	12	234123	chandan.shivaram12@gmail.com	123	✓

Archive MySQL database:



This design can be used in educational and any type of organization to validate the candidature.

7. BIBLIOGRAPHY

Weblinks

1. To learn about the software required to use, used, www.wikipidea.org.
2. For more examples for learning, referred, www.tutorialpoint.com.
3. For learning the Django, referred, www.w3schools.com., www.stackoverflow.com, www.developer.mozilla.org.