Implementing User Authentication and Authorization in Django:

Authentication vs Authorization

1. **Authentication**:
    - Authentication is the process of verifying the identity of a user.
    - The goal is to ensure that the user is who they say they are.
    - Typically, this involves the user providing credentials (like a username/email and password) which are checked against a stored set of data (e.g., a database).
2. **Authorization**:
    - Authorization is the process of granting or denying access to resources or actions based on the authenticated user's roles or permissions.
    - Once a user is authenticated, authorization determines what they can do within the application (e.g., view a specific page, edit data, or delete something).

User Authentication and Authorization in Django:

- Django provides a built-in authentication system inside the **django.contrib.auth** module, making it easy to secure your application.
- Ensure the following apps and middlewares are included in your settings.py:

```
INSTALLED_APPS = [

    ...

    'django.contrib.auth',  # Core authentication framework

    'django.contrib.contenttypes',  # Required for permissions

    'django.contrib.sessions',  # Manages sessions (critical for
logged-in users)

    'django.contrib.messages',  # For displaying messages (e.g.,
login success/failure)
]


MIDDLEWARE = [

    ...

    'django.contrib.sessions.middleware.SessionMiddleware',  #
Enables sessions

    'django.contrib.auth.middleware.AuthenticationMiddleware',  #
Associates users with requests

    'django.contrib.messages.middleware.MessageMiddleware',  #
Handles messages

]
```

- The **AuthenticationMiddleware** ties users to requests and enables the **request.user** object, which is essential for authentication checks.

User Model

- Django provides a default user model django.contrib.auth.models.User that contains fields like:
    - `username`
    - `password`
    - `email`
    - `first_name`,
    - `last_name`
    - `is_active`, (Boolean: determines if the user account is active)
    - `is_staff`, (Boolean: determines if the user can access Django Admin)
    - `is_superuser` (Boolean: determines if the user has all permissions)

Different between is_staff and is_superuser:

is_staff

- Determines whether a user can **access the Django admin panel**.
- If `is_staff = True`, the user can log in to the Django admin site.
- However, they **do not** have full permissions unless explicitly assigned.
- This user can log in to the Django admin panel but **cannot** modify objects unless given specific permissions.

is_superuser

- Grants **full access** to all parts of the Django admin site.
- If is_superuser = True, the user automatically has **all permissions**, regardless of what is assigned.
- This user has **complete control** over the admin panel and **all models**.

Example:

- Create a new django project called: **AuthProject1**

```
django-admin startproject AuthProject1
```

- Move inside the Project folder:

```
cd AuthProject1
```

- Perform the migrations (to create inbuilt application related tables)

```
python manage.py makemigrations

python manage.py migrate
```

- Verify the created tables inside the **db.sqlite3** database.
- Open the Django shell:

```
python manage.py shell
```

- Create a super user using the django shell:

```
from  django.contrib.auth.models import User

>>> user1= User.objects.create(username= 'ratan', email='ratan@gmail.com')

>>> user1.set_password('123')

>>> user1.is_superuser=True

>>> user1.is_staff=True

>>> user1.save()

>>>user1
```

- Here **set_password()** method will hash the password and store it inside the table
- Verify the **auth_user** table. (check the user created record inside the table)
- Perform the authentication in django shell:

```
>>> from django.contrib.auth import authenticate

>>> user = authenticate(username='ratan', password='123')

>>> user

<User: ratan>

>>> if user is not None:

            print('login successful')

else:

             print('invalid username or password')



>>> exit()
```

Example:

- Create a new Project called **ProductAuthProject**

  django-admin startproject ProductAuthProject

- Mode inside the Project folder

  cd ProductAuthProject

- Create a new application inside the project called **ProductApp**

  python manage.py startapp AuthApp

- Register the application inside the **settings.py** file.
- Perform the migrations to create the built-in applications related tables

  python manage.py makemigrations

  python manage.py migrate

- Define following view functions inside the **AuthApp/view.py** file

AuthApp/views.py

```
from django.shortcuts import render, redirect

from django.contrib import messages

from django.contrib.auth.models import User

from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.decorators import login_required

def home_view(request):
```

```python
    return render(request, 'home.html')


def register_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        cpassword = request.POST.get('cpassword')

        if password != cpassword:
            messages.error(request, 'Passwords do not match')
        elif User.objects.filter(username=username).exists():
            messages.error(request, 'Username already exists')
        elif User.objects.filter(email=email).exists():
            messages.error(request, 'Email already used')
        else:
            user = User.objects.create_user(
                username=username, email=email, password=password)
            messages.success(
                request, 'Registration successful! You can now login.')
            return redirect('login')
    return render(request, 'register.html')
```

```python
def login_view(request):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(request, username=username, password=password)

        if user:

            login(request, user)

            return redirect('dashboard')

        else:

            messages.error(request, 'Invalid username or password')

    return render(request, 'login.html',)


@login_required

def dashboard_view(request):

    return render(request, 'dashboard.html')


@login_required

def logout_view(request):

    logout(request)

    return redirect('home')
```

Explanation:

login(request, user):

- The `login()` function authenticates and logs in a user, creating a session for them.
- It stores the user's ID in the session.
- Sets `request.user` to the authenticated user object.
- This allows you to access `{{ user.username }}` in templates and `request.user` in views.
- `authenticate()` does not log the user in. It only verifies credentials. You must use `login()` to actually log them in.

logout(request):

- The `logout()` function ends the current user's session.
- Clears all session data related to the authenticated user.
- `request.user` becomes `AnonymousUser` again.

@login_required:

- The `@login_required` decorator protects a view so that only authenticated users can access it.
- If an unauthenticated user tries to access it, they will be redirected to the login page.

- **Specify the following statement inside the settings.py file.**

```
# Redirect to the url if any unauthenticated user tries to access
protected route

LOGIN_URL = 'login'
```

- Define the url patterns for the above view functions inside the **AuthApp/urls.py** file.

AuthApp/urls.py

```
from django.urls import path

from . import views

urlpatterns = [

    path('', views.home_view, name='home'),

    path('register/', views.register_view, name='register'),

    path('login/', views.login_view, name='login'),

    path('dashboard/', views.dashboard_view, name='dashboard'),

    path('logout/', views.logout_view, name='logout')

]
```

- **Include the above urls.py file at the project level urls.py file**

```
from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('AuthApp.urls'))

]
```

- Create the following HTML files inside the AuthApp/templates folder.

base.html:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>{% block title_block %} HTMLFormProject {% endblock %}</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.
css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6h
W+ALEwIH" crossorigin="anonymous">

    <style>

        body {

            margin: 0;

            padding: 0;

        }

        header {

            height: 10vh;

            background-color: rgb(139, 210, 26)

        }

        main {
            height: 80vh;

            background-color: aquamarine;

        }

        footer {

            height: 10vh;

            background-color: rgb(36, 188, 208);

        }

    </style>

</head>

<body>

    <header>

        <nav class="nav justify-content-end">

            <a class="nav-link" href="{% url 'home' %}">Product App</a>

            <a class="nav-link" href="{% url 'register' %}">Register</a>

            <a class="nav-link" href="{% url 'login' %}">Login</a>

            <a class="nav-link" href="{% url 'logout' %}">Logout</a>

        </nav>

    </header>

    <main class="overflow-auto">

        {% if messages %}

        {% for message in messages %}

        <div class="alert alert-warning  alert-dismissible fade show"
role="alert">

            {{message}}
            <button type="button" class="btn-close"
data-bs-dismiss="alert" aria-label="Close"></button>

        </div>

        {% endfor %}

            {% endif %}

        {% block main_block %}



        {% endblock %}

    </main>

    <footer>

        <p class="text-center">&copy; This is the footer section</p>

    </footer>



    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN
7N6jIeHz" crossorigin="anonymous"></script>

</body>

</html>
```

home.html

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

      {% block title_block %} Home Page {% endblock %}
      {% block main_block %}

      <h3 class="text-center">Welcome to Home page</h3>

      {% endblock %}
```

register.html

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

{% block title_block %} Registration Page {% endblock %}

{% block main_block %}

<h2 class="text-center">Register a User</h2>

<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label for="username" class="form-label">Enter
Username</label>

            <input type="text" name="username" id="username"
class="form-control" required>

        </div>

        <div class="mb-3">

            <label for="email" class="form-label">Enter Email</label>

            <input type="email" name="email" id="email"
class="form-control" required>

        </div>

        <div class="mb-3">
```

```
            <label for="password" class="form-label">Enter
Password</label>

            <input type="password" name="password" id="password"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="cpassword" class="form-label">Confirm
Password</label>

            <input type="password" name="cpassword" id="cpassword"
class="form-control" required>

        </div>

        <input type="submit" value="Register" class="btn btn-primary">

    </form>

    <p><a href="{% url 'login' %}">Already have an account? Login</a></p>

</div>

{% endblock %}
```

<u>login.html:</u>

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

{% block title_block %} Login Page {% endblock %}

{% block main_block %}

<h1 class="text-center">Login Page</h1>
```

```
<div class="container">
    <form method="POST">
        {% csrf_token %}
        <div class="mb-3">
            <label for="username" class="form-label">Enter
Username:</label>
            <input type="text" name="username" id="username"
class="form-control" required>
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Enter
Password:</label>
            <input type="password" name="password" id="password"
class="form-control" required>
        </div>
        <input type="submit" value="Login" class="btn btn-primary">
    </form>
    <p><a href="{% url 'register' %}">New User? Register</a></p>
</div>
{% endblock %}
```

dashboard.html

```
      {% extends 'base.html' %}
      {% block title_block %} Dashboard Page {% endblock %}
```

```
{% block main_block %}

<h1 class="text-center">Welcome to Dashboard</h1>

<h2>Welcome, {{ user.username }}</h2>


{% endblock %}
```

- Run the server and access the application:

  python manage.py runserver

  http://127.0.0.1:8000/

- Modify the header part of base.html file to show the links based on the user authentication:

base.html

```
 <header>

        <nav class="nav justify-content-end">

            <a class="nav-link" href="{% url 'home' %}">Product App</a>

            {% if user.is_authenticated %}

                <a class="nav-link" href="{% url 'dashboard'
%}">Dashboard</a>
```

```
                <a class="nav-link" href="{% url 'logout' %}">Logout</a>

            {% else %}

                <a class="nav-link" href="{% url 'register'
%}">Register</a>

                <a class="nav-link" href="{% url 'login' %}">Login</a>

            {% endif %}

        </nav>

    </header>
```

user.is_authenticated

- **user.is_authenticated** is a property of the **request.user** object in Django.
- It returns **True** if the current user is logged in, and **False** if the user is anonymous (not logged in).
- Django sets **`request.user`** for every request. If the user is
    - **logged in → `request.user` is a `User` object.**
    - **If not → it's an `AnonymousUser`.**

Managing Users via Django Admin:

- To manage all users in your Django application, you can create a superuser using the following command:

    python manage.py createsuperuser

- Once created, you can log in to the Django admin interface at:

    http://127.0.0.1:8000/admin/

- From the admin panel, you can:
    - View and manage all registered users
    - Edit user information (username, email, password, etc.)
    - Promote any user to an **admin** by setting:

- `is_staff = True` → Gives access to the admin panel
- `is_superuser = True` → Gives all permissions in the application

- **This interface is a powerful tool for admins to monitor, control, and customize user access across the application.**

Applying password validation:

- To Enforce the password validation inside the above application (applying inbuilt password validator)
- Modify the register_view function inside the **AuthApp/views.py** file as follows:

```
from django.contrib.auth.password_validation import validate_password

from django.core.exceptions import ValidationError

def register_view(request):

    if request.method == 'POST':



        username = request.POST.get('username')

        email = request.POST.get('email')

        password = request.POST.get('password')

        cpassword = request.POST.get('cpassword')



        if password != cpassword:

            messages.error(request, 'Passwords do not match')

        elif User.objects.filter(username=username).exists():

            messages.error(request, 'Username already exists')
        elif User.objects.filter(email=email).exists():
            messages.error(request, 'Email already used')
        else:
            try:
                validate_password(password)
                user = User.objects.create_user(
                    username=username, email=email, password=password)
                messages.success(
                    request, 'Registration successful! You can now
login.')
                return redirect('login')
            except ValidationError as e:
                for error in e:
                    messages.error(request, error)
    return render(request, 'register.html')
```

Implementing the User Authorization:

- Creating a functionality called view profile which can be visible and accessible only if the logged in user is super user(admin)

- **Modify the login.html to accept the role from the user:**

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

{% block title_block %} Login Page {% endblock %}

{% block main_block %}

<h1 class="text-center">Login Page</h1>

<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label for="username" class="form-label">Enter
Username:</label>

            <input type="text" name="username" id="username"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="password" class="form-label">Enter
Password:</label>

            <input type="password" name="password" id="password"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="role">Choose Role:</label>

            <select name="role" class="form-control" required>

                <option value="" disabled selected>Choose Role</option>
                <option value="user">User</option>

                <option value="admin">Admin</option>

            </select>

        </div>

        <input type="submit" value="Login" class="btn btn-primary">

    </form>

    <p><a href="{% url 'register' %}">New User? Register</a></p>

</div>

{% endblock %}
```

- **Modify the login_view function inside the AuthApp/view.py file as follows:**

```
def login_view(request):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        selected_role = request.POST.get('role')


        user = authenticate(request, username=username, password=password)

        if user:

            # Determine if the role matches the actual user status

            if (selected_role == 'admin' and user.is_superuser) or
(selected_role == 'user' and not user.is_superuser):
```

```
                login(request, user)

                return redirect('dashboard')

            else:

                messages.error(request, "Invalid credentials: Role
mismatch")

        else:

            messages.error(request, 'Invalid username or password')

    return render(request, 'login.html',)
```

- Define a view function inside the AuthApp/views.py file to render user_profile.html

```
@login_required

def view_profile(request):

    if not request.user.is_superuser:

        messages.error(request, "Access denied: You are not
authorized to view this page.")

        return redirect('dashboard')  # Redirect to dashboard or
home instead of showing raw 403


    return render(request, 'user_profile.html')
```

- Specify the url for the above view function inside the AuthApp/urls.py file:

```
path('view-profile/', views.view_profile, name='view_profile'),
```

- Create user_profile.html file inside AuthApp/templates folder

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

{% block title_block %} Profile Page {% endblock %}

{% block main_block %}

<h2>Superuser Profile</h2>

<p><strong>Username:</strong> {{ user.username }}</p>

<p><strong>Email:</strong> {{ user.email }}</p>

<p><strong>Is Superuser:</strong> {{ user.is_superuser }}</p>

<a href="{% url 'dashboard' %}" class="btn btn-secondary">Back to
Dashboard</a>

{% endblock %}
```

- Update the dashboard.html to include a View Profile link for the super user:

```
{% extends 'base.html' %}
{% block title_block %} Login Page {% endblock %}

{% block main_block %}

<h1 class="text-center">Welcome to Dashboard</h1>

<h2>Welcome, {{ user.username }}</h2>


{% if user.is_superuser %}

      <a href="{% url 'view_profile' %}" class="btn btn-info">View
Profile</a>

{% endif %}

{% endblock %}
```

Implementing User Specific Product Management: (CRUD Operations)

- Create another application inside the above ProductAuthProject with the name called ProductApp.

  python manage.py startapp ProductApp

- Register this ProductApp inside the settings.py file.
- Define the following mode class inside the ProductApp/models.py file

```
from django.db import models

from django.contrib.auth.models import User

class Product(models.Model):

    CATEGORY_CHOICES = [

        ('Electronics', 'Electronics'),

        ('Stationary', 'Stationary'),
        ('HomeApplience', 'HomeApplience'),

    ]

    # Specifying one to many relationship with User to Product

    user = models.ForeignKey(User, on_delete=models.CASCADE)

    name = models.CharField(max_length=100)

    price = models.IntegerField()

    quantity = models.IntegerField()

    category = models.CharField(max_length=20, choices=CATEGORY_CHOICES)

    created_at = models.DateTimeField(auto_now_add=True)

    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):

        return self.name

    class Meta:

        # ensures unique product name per user

        unique_together = ('user', 'name')
```

- Register the above model inside the **admin.py** file to manage products from the admin interface.

```
from django.contrib import admin

from .models import Product  # Import your Product model

class ProductAdmin(admin.ModelAdmin):

    list_display = ['id', 'name', 'price',
                    'quantity', 'category', 'user', 'created_at']

    list_filter = ['category', 'created_at']

admin.site.register(Product, ProductAdmin)
```

- **Run the server and login to the admin interface and add some products for a specific user.**

    python manage.py runserver

    http://127.0.0.1:8000/admin

- **Define the following view functions inside the ProductApp/views.py file**

```
from django.shortcuts import render

from ProductApp.models import Product

from django.contrib.auth.decorators import login_required

@login_required

def add_product_view(request):

   #  Fetch all products if superuser, else only their own

    if request.user.is_superuser:

        products = Product.objects.all()

    else:

        products = Product.objects.filter(user=request.user)

    return render(request, 'addproduct.html', context={'products':
products})
```

- **Create addproduct.html file inside the ProductApp/template folder.**

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

{% block title_block %} Add Product {% endblock %}

{% block main_block %}

<h2 class="text-center">Add a Product</h2>

<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label for="product_name" class="form-label">Enter
Product Name</label>

            <input type="text" name="product_name" id="product_name"
class="form-control" required>

        </div>

        <div class="mb-3">

            <label for="price" class="form-label">Enter
Price</label>

            <input type="number" name="price" id="price"
class="form-control" required>

        </div>

        <div class="mb-3">

            <label for="quantity" class="form-label">Enter
Quantity</label>
```

```
            <input type="number" name="quantity" id="quantity"
class="form-control" required>

        </div>

        <div class="mb-3">

            <label for="category">Choose Category:</label>

            <select name="category" class="form-control" required>

                <option value="" disabled selected>Choose
Category</option>

                <option value="Electronics">Electronics</option>

                <option value="Stationary">Stationary</option>

                <option value="HomeApplience">HomeApplience</option>

            </select>

        </div>

        <input type="submit" value="Add Product" class="btn
btn-success">

    </form>

</div>

{% if products %}

<h2 class="text-center">All Product Details</h2>

<hr>

<div class="container">

    <table class="table table-dark">

        <tr>

            <th>ProductId</th>
            <th>ProductName</th>

            <th>Price</th>

            <th>Quantity</th>

            <th>Category</th>

            <th>Created At</th>

            <th>Updated At</th>

            {% if user.is_superuser %}

            <th>Added By</th>

            {% endif %}

            <th>Action</th>

        </tr>


        {% for product in products %}


        <tr>

            <td>{{product.id}}</td>

            <td>{{product.name}}</td>

            <td>{{product.price}}</td>

            <td>{{product.quantity}}</td>

            <td>{{product.category}}</td>

            <td>{{product.created_at}}</td>

            <td>{{product.updated_at}}</td>

            {% if user.is_superuser %}
```

```
                <td>{{ product.user.username }}</td>
                {% endif %}
                <td>
                    <a href="#" class="btn btn-success
btn-sm">UPDATE</a>
                    <a href="#" class="btn btn-danger btn-sm">DELETE</a>
                </td>
            </tr>

            {% endfor %}

        </table>

    </div>
{% endif %}
{% endblock %}
```

- **Define the url pattern for the above** `add_product_view` **inside the ProductApp/urls.py file.**

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.add_product_view, name='addproduct')
]
```

- Include the above urls.py of the Product App inside the project level urls.py file.

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('AuthApp.urls')),
    path('products/', include('ProductApp.urls'))
]
```

- Modify the dashboard.html of AuthApp and include a link to manage the product.

```
{% extends 'base.html' %}
{% block title_block %} Login Page {% endblock %}
{% block main_block %}
<h1 class="text-center">Welcome to Dashboard</h1>
<h2>Welcome, {{ user.username }}</h2>

<a href="{% url 'addproduct' %}" class="btn btn-primary">Manage
Products</a>
{% if user.is_superuser %}
```

```
<a href="{% url 'view_profile' %}" class="btn btn-info">View
Profile</a>

{% endif %}

{% endblock %}
```

- **Run the server and access the application up to this point by logging as a user**

  python manage.py runserver

  http://127.0.0.1:8000/

- **Modify the the add_product_view to handle the add product form data**

```
from django.shortcuts import render, redirect

from ProductApp.models import Product

from django.contrib.auth.decorators import login_required

from django.contrib import messages

@login_required

def add_product_view(request):


    if request.method == 'POST':

        name = request.POST.get('product_name')

        price = request.POST.get('price')

        quantity = request.POST.get('quantity')

        category = request.POST.get('category')
        # Check if product already exists for this user

        if Product.objects.filter(user=request.user,
name=name).exists():

            messages.error(request, 'Product with this name already
exists.')

        else:

            Product.objects.create(

                user=request.user,

                name=name,

                price=price,

                quantity=quantity,

                category=category

            )

            messages.success(request, 'Product added successfully!')

            return redirect('addproduct')


    #  Fetch all products if superuser, else only their own

    if request.user.is_superuser:

        products = Product.objects.all()

    else:

        products = Product.objects.filter(user=request.user)

    return render(request, 'addproduct.html', context={'products':
products})
```

Implementing Delete Product functionality:

- **Define the following view function inside the ProductApp/views.py file**

```
@login_required

def delete_product_view(request, product_id):

    product = Product.objects.get(id=product_id)

    # Check if the product belongs to the logged-in user or if user
is superuser

    if request.user == product.user or request.user.is_superuser:

        product.delete()

        messages.success(request, 'Product deleted successfully!')

    else:

        messages.error(

            request, 'You are not authorized to delete this
product.')

    return redirect('addproduct')
```

- **Define the url pattern for the above view function inside ProductApp/urls.py file**

```
from django.urls import path

from . import views

urlpatterns = [

    path('', views.add_product_view, name='addproduct'),
    path('delete/<int:product_id>/',
         views.delete_product_view, name='deleteproduct'),
]
```

- Update the Delete Link in `addproduct.html`

```
<a href="{% url 'deleteproduct' product.id %}" class="btn btn-danger
btn-sm" onclick="return confirm('Are you sure you want to delete
this product?');">DELETE</a>
```

Implementing Update Product functionality:

- **Define the following view function inside the ProductApp/views.py file**

```
@login_required
def update_product_view(request, product_id):
    product = Product.objects.get(id=product_id)

    # Authorization: Only the product owner or a superuser can update
    if request.user != product.user and not request.user.is_superuser:
        messages.error(
            request, "You are not authorized to update this product.")
        return redirect('addproduct')
    if request.method == 'POST':
        product.name = request.POST.get('product_name')
        product.price = request.POST.get('price')
        product.quantity = request.POST.get('quantity')
        product.category = request.POST.get('category')

        # Ensure unique product name per user
        if Product.objects.filter(user=product.user,
name=product.name).exclude(id=product_id).exists():
            messages.error(request, "Product with this name already
exists.")
        else:
            product.save()
            messages.success(request, "Product updated successfully!")
            return redirect('addproduct')

    return render(request, 'updateproduct.html', {'product': product})
```

- **Define the url pattern for the above view function inside the ProductApp/urls.py file:**

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.add_product_view, name='addproduct'),
    path('delete/<int:product_id>/',
         views.delete_product_view, name='deleteproduct'),
    path('update/<int:product_id>/',
         views.update_product_view, name='updateproduct'),
]
```

- **Update the Update Link in `addproduct.html`**

```
<a href="{% url 'updateproduct' product.id %}" class="btn btn-success
btn-sm">UPDATE</a>
```

- **Create the updateproduct.html file inside ProductApp/templates folder**

```
{% extends 'base.html' %}
{% block title_block %}Update Product{% endblock %}

{% block main_block %}
<h2 class="text-center">Update Product</h2>
```

```
<div class="container">

    <form method="POST">

        {% csrf_token %}

        <div class="mb-3">

            <label class="form-label">Product Name</label>

            <input type="text" name="product_name" class="form-control"
value="{{ product.name }}" required>

        </div>

        <div class="mb-3">

            <label class="form-label">Price</label>

            <input type="number" name="price" class="form-control"
value="{{ product.price }}" required>

        </div>

        <div class="mb-3">

            <label class="form-label">Quantity</label>

            <input type="number" name="quantity" class="form-control"
value="{{ product.quantity }}" required>

        </div>

        <div class="mb-3">

            <label class="form-label">Category</label>

            <select name="category" class="form-control" required>

                <option value="Electronics" {% if product.category ==
"Electronics" %}selected{% endif %}>Electronics

                </option>
                <option value="Stationary" {% if product.category ==
"Stationary" %}selected{% endif %}>Stationary

                </option>

                <option value="HomeApplience" {% if product.category ==
"HomeApplience" %}selected{% endif %}>

                    HomeApplience</option>

            </select>

        </div>



        <input type="submit" value="Update Product" class="btn
btn-primary">

    </form>

</div>

{% endblock %}
```

Customizing the Inbuilt User model:

Why Customize the User Model?

Django's default User model is sufficient for many cases, but sometimes you need to add extra fields (like gender, phone number) or change the authentication method (e.g., using email instead of username). Customizing the User model allows you to:

- Add custom fields (e.g., name, gender, phone number).
- Use email as the unique identifier for authentication instead of a username.
- Extend functionality for superusers, staff, or regular users.
- Manage users through the Django admin panel.

Step 1: Create a new custom user model along with custom User Manager class inside the AuthApp/models.py file

```
from django.contrib.auth.models import AbstractUser, BaseUserManager

from django.db import models


class AppUserManager(BaseUserManager):

    def create_user(self, email, name,phone, address, gender,
password=None, **extra_fields):

        if not email:

            raise ValueError('The Email field is mandatory')

        # converting email with lower case

        email= self.normalize_email(email)

        user = self.model(

            email=email,

            name=name,

            phone=phone,

            address= address,

            gender= gender,

            **extra_fields

        )

        user.set_password(password)

        user.save(using=self._db)

        return user
    def create_superuser(self, email, name, phone, address, gender,
password=None, **extra_fields):

        extra_fields.setdefault('is_staff', True)

        extra_fields.setdefault('is_superuser', True)

        extra_fields.setdefault('is_active', True)


        if not password:

            raise ValueError("Superusers must have a password.")

        if extra_fields.get('is_staff') is not True:

            raise ValueError("Superuser must have is_staff=True.")

        if extra_fields.get('is_superuser') is not True:

            raise ValueError("Superuser must have is_superuser=True.")

        return self.create_user(email, name, phone, address, gender,
password, **extra_fields)


class AppUser(AbstractUser):


    GENDER_CHOICES = (

        ('Male', 'Male'),

        ('Female', 'Female'),

        ('Other', 'Other'),

    )

    # redefine the email to be unique
    email = models.EmailField(unique=True)


    # Adding extra field  inside the custom user class

    name = models.CharField(max_length=100)

    phone = models.CharField(max_length=15)

    address = models.TextField()

    gender = models.CharField(max_length=10, choices=GENDER_CHOICES)

    # removing the username field

    username= None

    objects = AppUserManager()

    USERNAME_FIELD = 'email'

    REQUIRED_FIELDS = ['name', 'phone', 'address', 'gender']


    def __str__(self):

        return self.email
```

Explanation:

`AppUserManager` (Custom User Manager)

This class inherits from `BaseUserManager`, which provides helper methods to manage user creation.

`create_user(...)`

- This method creates a regular user.
- Check if an email is provided.

- Normalizes the email (lowercase).
- Fills in required fields (`email`, `name`, `phone`, `address`, `gender`).
- Sets the password securely using `set_password`.
- Saves the user to the database.
- Inside a custom manager like `BaseUserManager`, `self._db` refers to the **current database connection** being used by the manager.

  This is especially useful when you're:

  - Using **multiple databases** in your Django project.
  - Ensuring compatibility with Django's **multi-database routing system**.

`create_superuser(...)`

- This method creates a superuser with additional privileges.
- Ensures `is_staff`, `is_superuser`, and `is_active` are set to `True`.
- Validates that a password is provided.
- Calls `create_user` internally with these fields.

Why a custom manager? Django needs it to properly handle `createsuperuser` with `email` as the unique field.

`AppUser` (Custom User Model)

This class inherits from `AbstractUser` to retain useful fields like:

- `is_active`, `is_staff`, `is_superuser`
- Password and group/permissions handling

Customizations:

- `username = None`

    - **Removes default `username` field** from `AbstractUser`.

- `email = models.EmailField(unique=True)`
    - Makes `email` the unique identifier.

`USERNAME_FIELD = 'email'`

- Tells Django to use `email` as the login identifier.

`REQUIRED_FIELDS = [...]`

- Fields required when creating a superuser using the command line.

`objects = AppUserManager()`

- Uses your custom manager for user creation logic.

Step2: Specify the above created AppUser model class inside the settings.py file to tell Django to use this AppUser class instead of using default User class.

<u>settings.py file:</u>

```
    AUTH_USER_MODEL = 'AuthApp.AppUser'
```

Step3: Modify the model class inside the ProductApp/models.py file to refer to the AppUser instead of using the User class.

```
from django.conf import settings

 user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
```

Step 4: Perform the migrations

python manage.py makemigrations

python manage.py migrate

Step5: Verify the created tables inside the **db.sqlite3** database

Step6: Create a **superuser** from the command prompt:

python manage.py createsuperuser

Step7: Register **AppUser** class along with its Admin class inside the **AuthApp/admin.py** file to manage the users from the admin interface:

```
from django.contrib import admin

from django.contrib.auth.admin import UserAdmin

from .models import AppUser


class AppUserAdmin(UserAdmin):

    model = AppUser

    list_display = ('email', 'name', 'is_staff', 'is_superuser', 'gender',
'address', 'phone')

    ordering = ('email',)


    fieldsets = (

        (None, {'fields': ('email', 'password')}),
        ('Personal Info', {'fields': ('name', 'phone', 'address',
'gender')}),

        ('Permissions', {'fields': ('is_active', 'is_staff',

         'is_superuser', 'groups', 'user_permissions')}),

        ('Important dates', {'fields': ('last_login',)}),



    )



    )

    # Fields visible when creating a new user

    add_fieldsets = (

        (None, {

            'classes': ('wide',),

            'fields': ('email', 'name', 'phone', 'address', 'gender',
'password1', 'password2'),

        }),

    )



admin.site.register(AppUser, AppUserAdmin)
```

Step8: Run the server and access the admin interface to manage the users.

```
python manage.py runserver
```

http://127.0.0.1:8000/admin/

Step9: Modify the **register.html** file and **login.html** inside the **AuthApp/templates** folder for the registration and login.

register.html:

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

{% block title_block %} Registration Page {% endblock %}

{% block main_block %}

<h2 class="text-center">Register a User</h2>

<div class="container">

    <form method="POST">

        {% csrf_token %}


        <div class="mb-3">

            <label for="name" class="form-label">Enter Full Name</label>

            <input type="text" name="name" id="name" class="form-control" 
required>

        </div>


        <div class="mb-3">

            <label for="email" class="form-label">Enter Email</label>

            <input type="email" name="email" id="email" 
class="form-control" required>

        </div>
        <div class="mb-3">

            <label for="phone" class="form-label">Enter Phone
Number</label>

            <input type="text" name="phone" id="phone"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="address" class="form-label">Enter Address</label>

            <textarea name="address" id="address" class="form-control"
rows="3" required></textarea>

        </div>



        <div class="mb-3">

            <label for="gender" class="form-label">Select Gender</label>

            <select name="gender" id="gender" class="form-select"
required>

                <option value="" disabled selected>--Select
Gender--</option>

                <option value="Male">Male</option>

                <option value="Female">Female</option>

                <option value="Other">Other</option>

            </select>

        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Enter
Password</label>
            <input type="password" name="password" id="password"
class="form-control" required>
        </div>

        <div class="mb-3">
            <label for="cpassword" class="form-label">Confirm
Password</label>
            <input type="password" name="cpassword" id="cpassword"
class="form-control" required>
        </div>

        <input type="submit" value="Register" class="btn btn-primary">
    </form>

    <p class="mt-3">
        <a href="{% url 'login' %}">Already have an account? Login</a>
    </p>
</div>
{% endblock %}
```

login.html:

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

{% block title_block %} Login Page {% endblock %}

{% block main_block %}

<h1 class="text-center">Login Page</h1>

<div class="container">

    <form method="POST">

        {% csrf_token %}



        <div class="mb-3">

            <label for="email" class="form-label">Enter Email:</label>

            <input type="email" name="email" id="email"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="password" class="form-label">Enter
Password:</label>

            <input type="password" name="password" id="password"
class="form-control" required>

        </div>



        <div class="mb-3">

            <label for="role" class="form-label">Choose Role:</label>
            <select name="role" class="form-control" required>

                <option value="" disabled selected>Choose Role</option>

                <option value="user">User</option>

                <option value="admin">Admin</option>

            </select>

        </div>


        <input type="submit" value="Login" class="btn btn-primary">

    </form>


    <p class="mt-3">

        <a href="{% url 'register' %}">New User? Register</a>

    </p>

</div>

{% endblock %}
```

Step10: Modify the **register_view** function inside the **AuthApp/views.py** file accordingly.

```
from django.contrib.auth import get_user_model

User = get_user_model()  # Use your custom user model

def register_view(request):

    if request.method == 'POST':
```

```python
name = request.POST.get('name')

email = request.POST.get('email')

phone = request.POST.get('phone')

address = request.POST.get('address')

gender = request.POST.get('gender')

password = request.POST.get('password')

cpassword = request.POST.get('cpassword')


if password != cpassword:

    messages.error(request, 'Passwords do not match')

elif User.objects.filter(email=email).exists():

    messages.error(request, 'Email already registered')

else:

    try:

        validate_password(password)

        User.objects.create_user(

            email=email,

            name=name,

            phone=phone,

            address=address,

            gender=gender,

            password=password

        )
```

```
                messages.success(request, 'Registration successful! You
can now login.')
                return redirect('login')
            except ValidationError as e:
                for error in e:
                    messages.error(request, error)
    return render(request, 'register.html')
```

- Modify the **login_view** function inside the **AuthApp/views.py** file accordingly

```
def login_view(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')
        selected_role = request.POST.get('role')

        user = authenticate(request, email=email, password=password)
        if user:
            # Determine if the role matches the actual user status
            if (selected_role == 'admin' and user.is_superuser) or
(selected_role == 'user' and not user.is_superuser):
                login(request, user)
                return redirect('dashboard')
            else:

                messages.error(request, "Invalid credentials: Role
mismatch")

        else:

            messages.error(request, 'Invalid username or password')

    return render(request, 'login.html',)
```

- Update the **dashboard.html** file

```
<h2>Welcome, {{ user.name }}</h2>
```

- Update the **user_profile.html** file:

```
<p><strong>Username:</strong> {{ user.name }}</p>
```

- Update the **addproduct.html** file:

```
<td>{{ product.user.name }}</td>
```