Development of Signup and Dashboard Functionality Using Django

https://github.com/Poorani-27/Django_signup_form.git

Poorani T

Tpoorani2002@gmail.com

1. Introduction

The project focuses on developing a robust signup and dashboard functionality using Django, aimed at providing users with a seamless registration process and an intuitive dashboard experience. This report outlines the implementation details, challenges encountered, and solutions applied during the development phase. The project utilizes Django for backend logic, MySQL for data management, and Bootstrap for frontend design to ensure scalability and user-friendly interfaces.

2. Project Setup and Configuration

2.1 Setting up Django Environment

Python 3.7+ and Django 3.2 were chosen for their stability and extensive community support. Virtual environments were used to isolate dependencies, ensuring a clean setup across different development environments.

2.2 Project Requirements

The project requirements centered around developing a web application using Django, focusing on creating a user signup and login system for both patients and doctors. Key features included secure password handling, profile management with image uploads, and distinct dashboards for each user type. The application aimed to provide a seamless user experience while ensuring data security and efficient database management.

2.3 Frontend Design Considerations

Bootstrap was integrated to streamline UI development, ensuring responsiveness and visual appeal across devices. Custom CSS styles were applied to enhance the user interface (UI), maintaining Django’s templating system for dynamic content rendering.

3. Implementation Details

3.1 Signup Functionality

Users can register as patients or doctors through a streamlined signup process. The form includes fields for name, email, address, profile photo, username, and password. Password validation rules enforce complexity requirements, while Django's file handling manages profile photo uploads.

3.2 Dashboard Functionality

Upon successful signup and login, users are redirected to personalized dashboards. Patients and doctors view tailored information such as name, email, and specific user-type details. The dashboard features include logout and account management options for user convenience.

4. Technical Challenges and Solutions

4.1 Password Validation Implementation

Implementing strong password validation required custom rules using regular expressions (regex) to ensure secure user credentials. Django’s password validation settings were adjusted to meet project-specific security standards.

4.2 Handling User Types in Signup Form

Managing user types (patient or doctor) within the signup form involved conditional logic and model inheritance to differentiate data entries. This approach ensured accurate data storage and user segmentation.

4.3 Integration of Django Templating and Forms

Django forms were seamlessly integrated into HTML templates using template tags and context variables. Real-time validation feedback improved user experience by guiding input accuracy and reducing errors.

5. User Experience and Interface Design

5.1 UX Optimization in Signup Process

The signup process prioritized user-friendliness with clear instructions and error handling. Responsive design principles facilitated optimal usability on various devices, enhancing accessibility and user satisfaction.

5.2 Dashboard Usability and Navigation

The dashboard interface offered intuitive navigation and access to essential user information. Simple layouts and prominent action buttons ensured easy interaction and efficient task completion for users.

6. Conclusion and Future Enhancements

6.1 Summary of Achievements

The project successfully implemented signup and dashboard functionalities using Django, achieving secure user registration and personalized dashboard experiences. Key accomplishments included robust password validation, seamless form integration, and responsive UI design.

6.2 Lessons Learned

Key insights included the importance of meticulous validation, modular code structure, and iterative UI improvements based on user feedback. Collaboration and communication within the development team were crucial for overcoming challenges and achieving project goals.

6.3 Future Enhancements

Future developments may include advanced user profile management features, integration with external APIs for extended functionality, and performance optimizations to handle increased user traffic and data processing demands.

7. Appendix

7.1 Code Snippets

Excerpts from Django views, models, and templates showcasing key implementation details and customization.

Models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):
    USER_TYPE_CHOICES = (
        ('patient', 'Patient'),
        ('doctor', 'Doctor'),
    )

    user_type = models.CharField(max_length=10, choices=USER_TYPE_CHOICES,
default='patient')
    is_patient = models.BooleanField(default=False)
    is_doctor = models.BooleanField(default=False)

    groups = models.ManyToManyField(
        'auth.Group',
        related_name='customuser_set',
        blank=True,
        help_text='The groups this user belongs to. A user will get all
permissions granted to each of their groups.',
        verbose_name='groups'
    )
    user_permissions = models.ManyToManyField(
        'auth.Permission',
        related_name='customuser_set',
        blank=True,
        help_text='Specific permissions for this user.',
        verbose_name='user permissions'
    )

    def __str__(self):
        return self.username

class Patient(models.Model):
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    profile_picture = models.ImageField(upload_to='profile_pics/')
    email = models.EmailField()
    address_line1 = models.CharField(max_length=255)
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    pincode = models.CharField(max_length=10)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

class Doctor(models.Model):
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    profile_picture = models.ImageField(upload_to='profile_pics/')
    email = models.EmailField()
    address_line1 = models.CharField(max_length=255)
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    pincode = models.CharField(max_length=10)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

views.py

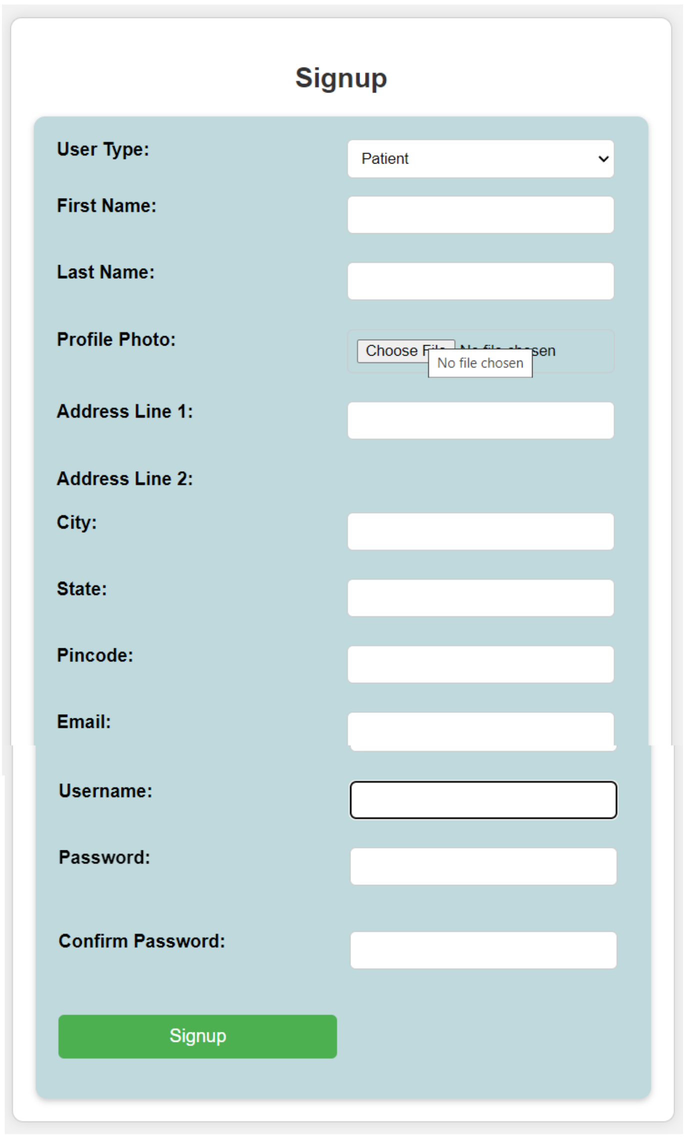
```
# accounts/views.py

from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.contrib.auth import login
from django.core.exceptions import ObjectDoesNotExist
from .forms import CustomUserCreationForm, PatientSignupForm, DoctorSignupForm
```

```python
from .models import CustomUser, Patient, Doctor

@login_required
def dashboard(request):
    user = request.user
    return render(request, 'accounts/dashboard.html', {'user': user})

def signup(request):
    if request.method == 'POST':
        user_form = CustomUserCreationForm(request.POST, request.FILES)
        patient_form = PatientSignupForm(request.POST, request.FILES) if 'is_patient' in request.POST else PatientSignupForm()
        doctor_form = DoctorSignupForm(request.POST, request.FILES) if 'is_doctor' in request.POST else DoctorSignupForm()

        if user_form.is_valid():
            user = user_form.save(commit=False)
            user.user_type = 'patient' if 'is_patient' in request.POST else 'doctor'
            if user.user_type == 'patient':
                if patient_form.is_valid():
                    patient = patient_form.save(commit=False)
                    patient.user = user
                    patient.save()
            elif user.user_type == 'doctor':
                if doctor_form.is_valid():
                    doctor = doctor_form.save(commit=False)
                    doctor.user = user
                    doctor.save()
            user.save()
            login(request, user)
            return redirect('account_created')  # Redirect to account created success page
    else:
        user_form = CustomUserCreationForm()
        patient_form = PatientSignupForm()
        doctor_form = DoctorSignupForm()

    return render(request, 'accounts/signup.html', {
        'user_form': user_form,
        'patient_form': patient_form,
        'doctor_form': doctor_form
    })

def account_created(request):
    return render(request, 'accounts/account_created.html')  # Render success page
```

7.2 Screenshots

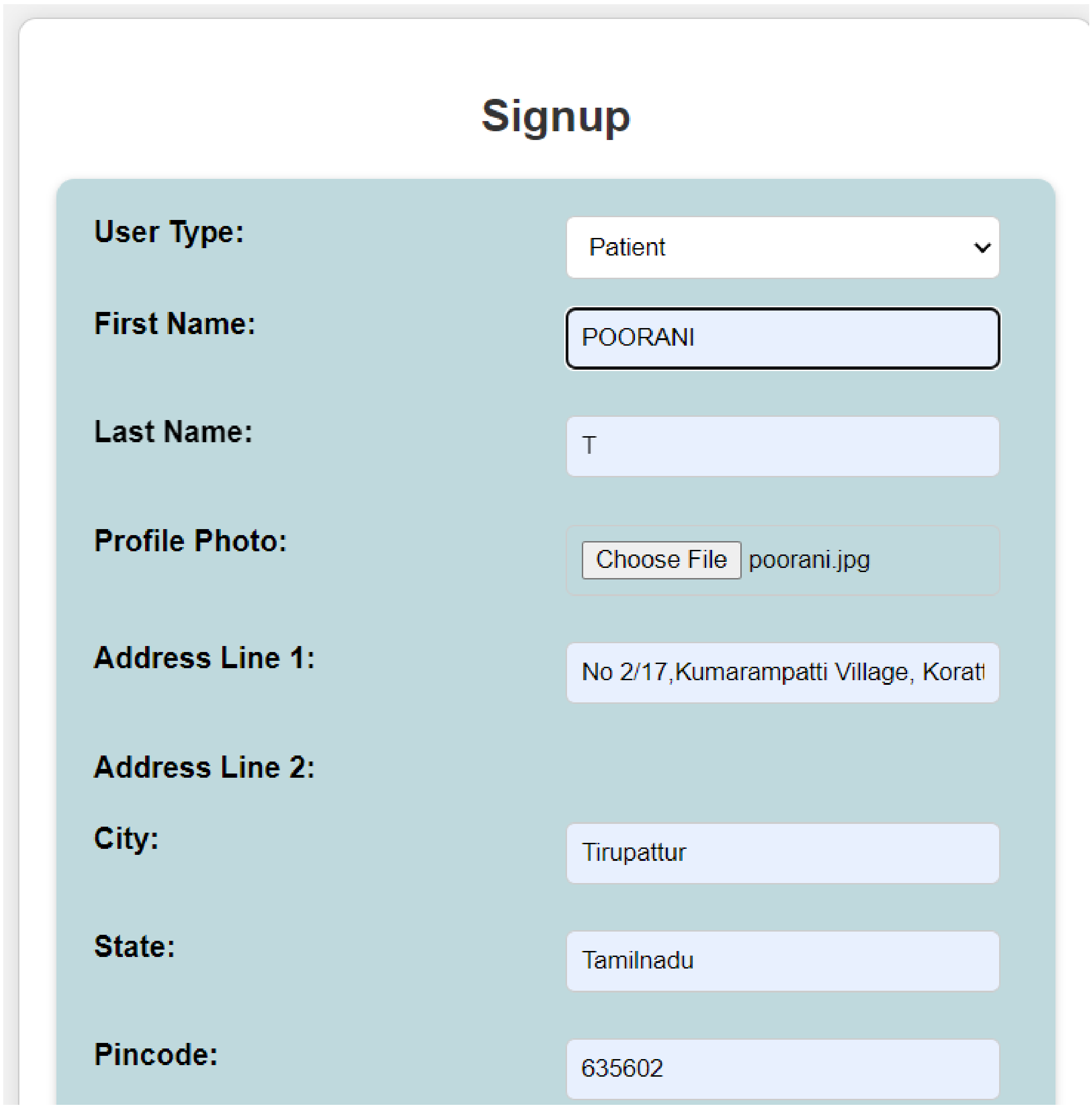
Visual representations of signup forms, dashboard interfaces, and responsive design elements.

Signup Form



Signup

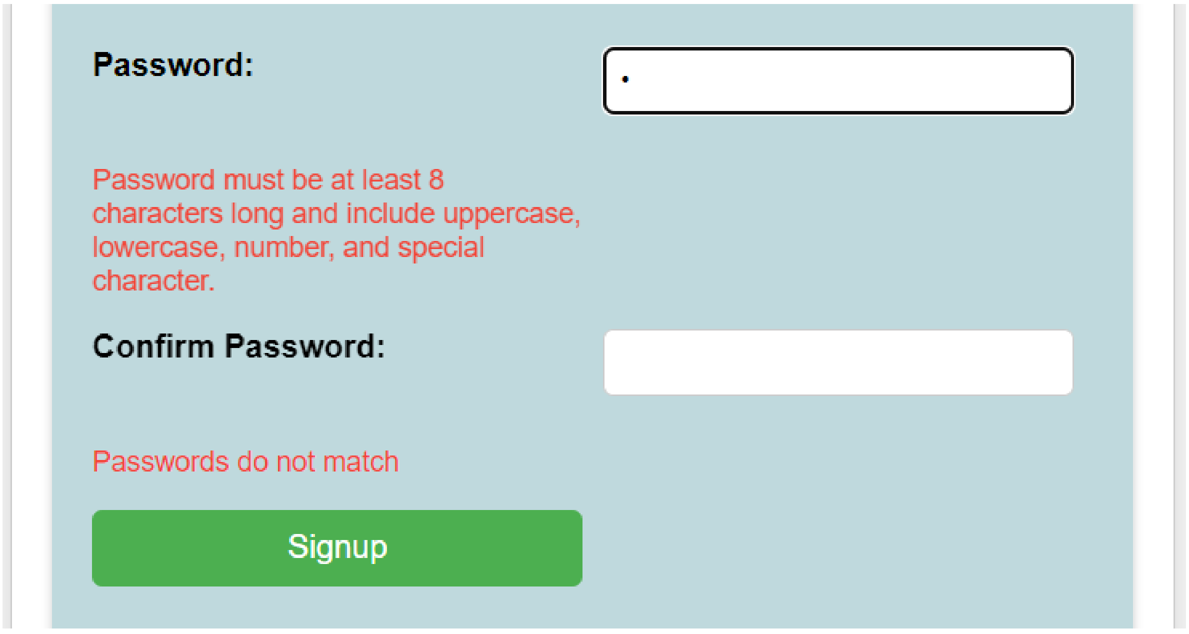
User Type:	Patient
First Name:	
Last Name:	
Profile Photo:	Choose File No file chosen
Address Line 1:	
Address Line 2:	
City:	
State:	
Pincode:	
Email:	
Username:	
Password:	
Confirm Password:	



Signup

User Type:	Patient
First Name:	POORANI
Last Name:	T
Profile Photo:	Choose File poorani.jpg
Address Line 1:	No 2/17,Kumarampatti Village, Koratt
Address Line 2:	
City:	Tirupattur
State:	Tamilnadu
Pincode:	635602

Form With the necessary fields including FirstName, LastName, Profile photo And Full Address

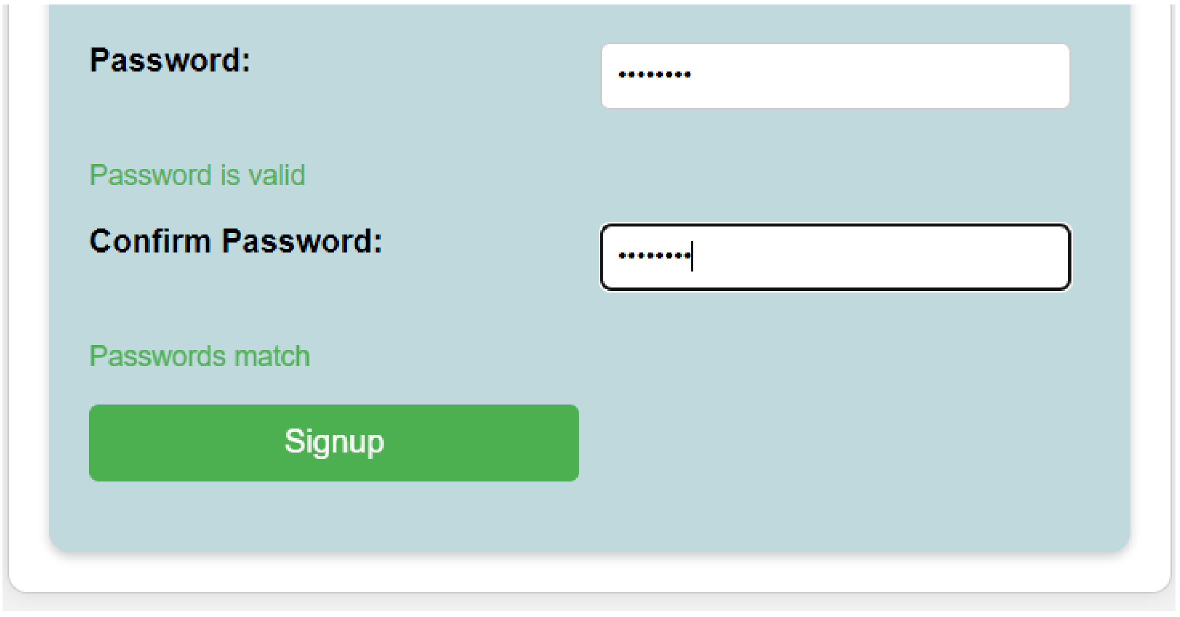

Password:

Password must be at least 8 characters long and include uppercase, lowercase, number, and special character.

Confirm Password:

Passwords do not match

Password Requirements

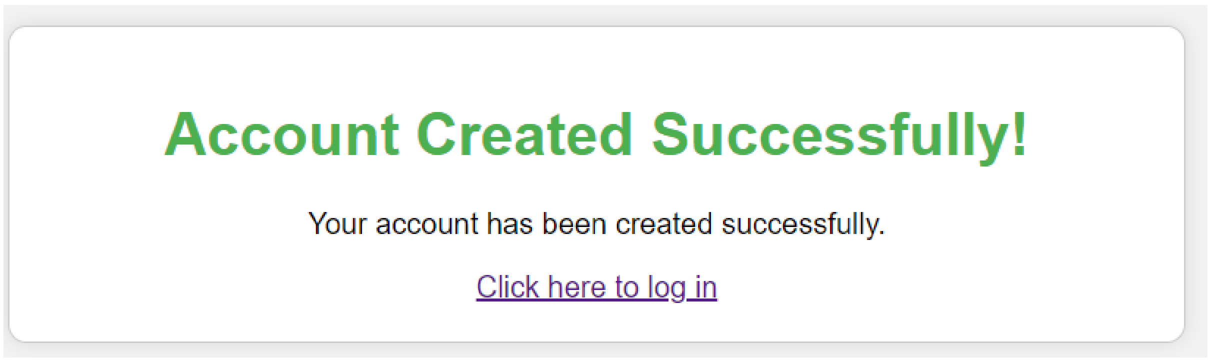

Password:

Password is valid

Confirm Password:

Passwords match

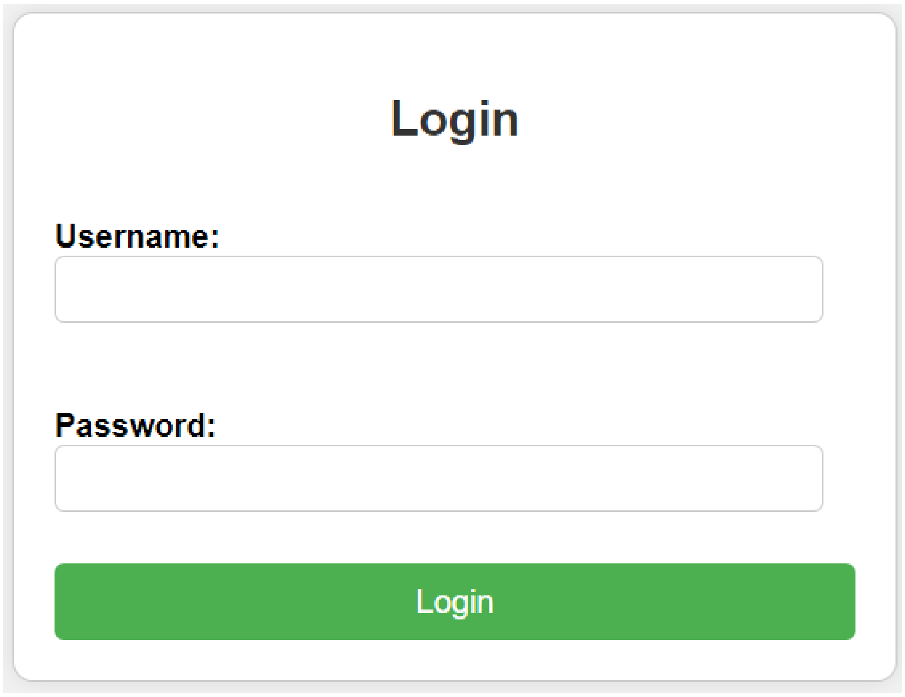
Password Validation


Account Created Successfully!

Your account has been created successfully.

Click here to log in

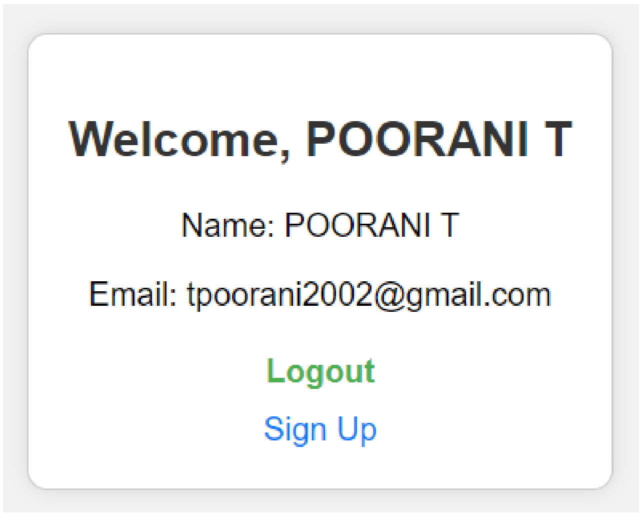
Account Creation Successful


Login

Username:

Password:

Login Page


Welcome, POORANI T

Name: POORANI T

Email: tpoorani2002@gmail.com

Dashboard