DJANGO USER REGISTRATION WITH EMAIL CONFIRMATION

-NAME: BUDUMURU YUGANDHAR

- ROLL NO: 21B91A0426

- COLLEGE: SRKR Engineering College

- FULL STACK PYTHON INTERNSHIP

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Project Overview

The Django User Registration System is a web application that allows users to register, login, and manage their profiles. This system includes features such as email confirmation, password reset, and user profile management.

1.2 Objectives

- To develop a user-friendly registration system.

- To ensure secure user authentication.

- To implement robust backend functionality using Django.

- To integrate frontend and backend seamlessly.

1.3 Scope

This document covers the steps for setting up the development environment, initializing the project, integrating frontend code, creating models, views, and forms, setting up Django admin, and running the development server.

2. SETUP DEVELOPMENT ENVIRONMENT

2.1 Install Necessary Software

- Install Python from [python.org](https://www.python.org/downloads/).
- Install Django using pip:

- pip install django

2.2 Set Up Integrated Development Environment (IDE)

- Install an IDE like Visual Studio Code, PyCharm, or IntelliJ IDEA.

3. PROJECT INTIALIZATION

3.1 Create Project Directory

mkdir user_registration_integration

cd user_registration_integration

3.2 Initialize Version Control System

git init

3.3 Create Django Project

django-admin startproject user_registration .

3.4 Create Django App for Accounts

python manage.py startapp accounts

4. INTEGRATE FRONTEND CODE

4.1 Move Frontend Files

Move HTML, CSS, and JavaScript files into the `accounts` app's `templates` and `static` directories.

4.2 Configure Static and Templates

In `settings.py`:

```python
import os

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'accounts/static'),
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'accounts/templates')],
```

4.1 Move Frontend Files

Move HTML, CSS, and JavaScript files into the `accounts` app's `templates` and `static` directories.

4.2 Configure Static and Templates

In `settings.py`:

```python
import os

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'accounts/static'),
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'accounts/templates')],
```

```
'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
  },
]
```

4.3 Update URL Patterns

In `user_registration/urls.py`:

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls')),
```

```
'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
  },
]
```

4.3 Update URL Patterns

In `user_registration/urls.py`:

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls')),
```

]
```

In `accounts/urls.py`:

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

urlpatterns = [
    path('register/', views.register, name='register'),
]
```

```
]
```

In `accounts/urls.py`:

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

urlpatterns = [
    path('register/', views.register, name='register'),
]
```

---

5. CREATE MODELS, VIEWS, AND FORMS

5.1 Create Models

In `accounts/models.py`:

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

class CustomUser(AbstractUser):
    email_confirmed = models.BooleanField(default=False)
```

5.2 Create Forms

In `accounts/forms.py`:

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import CustomUser
```

class CustomUserCreationForm(UserCreationForm):

    class Meta:

5. CREATE MODELS, VIEWS, AND FORMS

5.1 Create Models

In `accounts/models.py`:

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


class CustomUser(AbstractUser):
    email_confirmed = models.BooleanField(default=False)
```

5.2 Create Forms

In `accounts/forms.py`:

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import CustomUser
```

```python
class CustomUserCreationForm(UserCreationForm):
    class Meta:
```



```python
        model = CustomUser
        fields = ('username', 'email', 'password1', 'password2')
```

5.3 Create Views

In `accounts/views.py`:

```python
from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate
from .forms import CustomUserCreationForm

def register(request):
    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            raw_password = form.cleaned_data.get('password1')
            user = authenticate(username=user.username, password=raw_password)
            login(request, user)
            return redirect('home')
```

```python
    else:

        form = CustomUserCreationForm()

    return render(request, 'accounts/register.html', {'form': form})
```



```

## 5.4 Create HTML Template

In `accounts/templates/accounts/register.html`:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
    <title>Register</title>
</head>
<body>
    <div class="container">
        <h2>Register</h2>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">Register</button>
```

```
    </form>

  </div>

  <script src="{% static 'js/main.js' %}"></script>
```



```
```

5.4 Create HTML Template

In `accounts/templates/accounts/register.html`:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{% static 'css/styles.css' %}">
  <title>Register</title>
</head>
<body>
  <div class="container">
    <h2>Register</h2>
    <form method="post">
      {% csrf_token %}
      {{ form.as_p }}
      <button type="submit">Register</button>
```
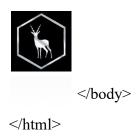
```
      </form>

    </div>

    <script src="{% static 'js/main.js' %}"></script>
```



```
            </body>

</html>
```

6. CREATE DJANGO ADMIN AND SUPERUSER

6.1 Register Models in Admin

In `accounts/admin.py`:

```python
from django.contrib import admin
from .models import CustomUser
from django.contrib.auth.admin import UserAdmin


admin.site.register(CustomUser, UserAdmin)
```

6.2 Create and Apply Migrations

```sh
python manage.py makemigrations
```

python manage.py migrate

```

## 6.3 Create Superuser



python manage.py createsuperuser

## 7. RUN THE DEVELOPMENT SERVER

## Start the Django development server:

```sh
python manage.py runserver
```

Visit `http://localhost:8000/accounts/register/` to see the integrated frontend and backend registration form.

8. TESTING

8.1 Unit Testing

- Create test cases for models, forms, and views.

- Ensure that the user registration process works as expected.

- Test email confirmation and password reset functionalities.

8.2 Integration Testing

- Test the integration between the frontend and backend.

- Verify that the static files are served correctly.

- Ensure that the user registration form submits data to the backend properly.

8.3 User Acceptance Testing

- Gather feedback from users to ensure the registration system is user-friendly.

- Make adjustments based on user feedback to improve the user experience.

9. Conclusion

The Django User Registration System project demonstrates the integration of frontend and backend components to create a functional user registration system. By following the steps outlined in this document, developers can set up a secure and user-friendly registration system using Django.

9.1 Future Enhancements

- Implement email sending for confirmation and password reset.

- Improve the user interface for better user experience.

- Add more functionalities such as user profile management and social media login integration.