Advanced Django Exercise: User Management and Authentication System

This exercise will focus on creating a **User Management and Authentication System** using Django. Students will explore how to use Django's forms, validation, user management features, and authentication. The goal is to build a system where users can register, log in, and manage their profiles while also understanding how to customize and extend Django's user model.

Exercise Overview

You will build a user management system with the following features:

1. **User Registration**: Users can sign up for an account using a custom registration form.
2. **Login/Logout**: Users can log in and out using Django's built-in authentication system.
3. **Profile Management**: Authenticated users can manage their profile, update their information, and change their password.
4. **Form Validation**: Implement custom form validation to ensure data integrity.
5. **Custom User Model**: Extend or substitute Django's default `User` model.
6. **Authentication Backend**: Write a custom authentication backend to handle user authentication in a custom way.

Tasks

1. Project Setup and Configuration

- Create a new Django project called `user_management`.
- Create an app called `accounts`.

---

2. Create and Manage Users in Django

1. **Custom User Model**:
    - Extend or substitute Django's default `User` model.
    - Add fields like `phone_number`, `date_of_birth`, or `profile_picture`.
2. **ModelForm**:
    - Use Django's **ModelForm** to create a registration form that captures user details (username, email, password) along with the custom fields you've added.

---

3. User Registration

1. **Form Creation**:
    - Build a **registration form** using Django's **ModelForm**.
    - Add custom validation for fields like `phone_number` (e.g., ensuring it contains only digits and has a specific length).
2. **Form and Field Validation**:
    - Implement custom validation methods for the form (e.g., check if the email already exists, password complexity).
    - Use **Django Forms Capabilities** to leverage automatic form handling and validation.
3. **View for User Registration**:
    - Create a **view** that handles both GET and POST requests to display the registration form and process form submissions.
    - Ensure data is validated and saved using the **ModelForm**.

---

4. Login and Logout URLs and Views

1. **Login**:
    - Use Django's built-in `LoginView` to handle user login.
    - Add a `login.html` template where users can enter their credentials.
2. **Logout**:
    - Use Django's built-in `LogoutView` to handle user logout.
    - Create a `logged_out.html` page that informs users they have successfully logged out.
3. **Login Required**:
    - Protect specific views (e.g., profile management page) using Django's `login_required` decorator.

---

5. Profile Management

1. **Profile Page**:
    - Create a view that displays the logged-in user's information.
    - Allow users to update their profile information, such as email, phone number, and profile picture.
2. **Password Change**:
    - Use Django's built-in **password change view** to allow users to update their passwords.
3. **Form for Profile Update**:
    - Create a **form** that extends the user's profile and allows for field validation on updates (e.g., phone number validation, email uniqueness check).

---

6. Writing a Custom Authentication Backend

1. **Custom Authentication Backend**:
    - Write a custom authentication backend that allows users to log in using either their **username or email**.
    - Implement the backend by extending `BaseBackend` and overriding the `authenticate` method.
2. **Configure the Backend**:
    - Add the custom authentication backend to `settings.py` so that Django uses it for user authentication.

---

7. Login and Views

1. **Custom Login View**:
    - Implement a custom login view that uses the custom authentication backend.
    - Create a `login.html` template with fields for username/email and password.
2. **Login Redirect**:
    - After login, redirect the user to their profile page or a dashboard.

Bonus Challenges

1. **Email Confirmation**:
    - After user registration, send a confirmation email to the user's email address with a link to verify their account.
2. **Password Reset**:
    - Implement Django's built-in password reset functionality with email support.
3. **Social Authentication**:
    - Integrate social authentication using third-party providers like Google or Facebook.

What Students Should Submit:

- A fully functional Django project with the `user_management` project and `accounts` app.
- Working registration, login, and profile management views.
- Proper form validation, custom user model, and authentication backend.
- Protected views using Django's authentication system.

Learning Outcomes:

- Practice using Django's **forms** and **ModelForm** to build user registration and profile management systems.
- Learn to implement **custom validation** and extend built-in form capabilities.
- Understand Django's **authentication system**, including login, logout, and password management.
- Learn how to create a **custom authentication backend** to extend Django's default behavior.
- Gain experience with extending or substituting Django's **User model** to add custom fields and functionality.