Steps to Enable Deakin Email-based Signup with Secure Password Validation

This document outlines the changes made to enable a signup process that validates Deakin email addresses, ensures strong password policies, and hashes passwords securely. Follow these steps to implement this functionality in your project.

## 1. Changes to sign-up.html

**Location**: home/templates/accounts/sign-up.html

**Old Code:**

* The old form did not validate email addresses in real time and lacked detailed feedback for passwords.

**New Code:**

1. **Added Deakin Email Validation with Real-Time Feedback**:

{% if field.name == 'email' %}

<span id="email-error" class="text-danger"></span>

{% elif field.name == 'password1' %}

<span id="password-error" class="text-danger"></span>

{% endif %}

1. **JavaScript for Real-Time Validation**:

document.addEventListener("DOMContentLoaded", function() {

const emailField = document.getElementById("id\_email");

const emailErrorMessage = document.getElementById("email-error");

const emailPattern = /@deakin\.edu\.au$/;

emailField.addEventListener("input", function() {

const emailValue = emailField.value.trim();

if (emailPattern.test(emailValue)) {

emailErrorMessage.textContent = "";

emailField.classList.remove("is-invalid");

emailField.classList.add("is-valid");

} else {

emailErrorMessage.textContent = "Email must match your Deakin email address";

emailField.classList.remove("is-valid");

emailField.classList.add("is-invalid");

}

});

});

1. **Enhanced Password Validation**: Added a real-time password requirements checklist to ensure compliance with the policy.

The system provides live feedback using a checklist of requirements such as:

* **Length**: At least 8 characters.
* **Lowercase Letter**: At least one lowercase letter.
* **Uppercase Letter**: At least one uppercase letter.
* **Number**: At least one numeric digit.
* **Special Symbol**: At least one special character (e.g., @, $, %, &).

## 2. Changes to forms.py

**Location**: home/forms.py

**Old Code:**

* The email validation was too lenient, allowing non-Deakin email addresses.
* Password validation was missing strong requirements.

**New Code:**

1. **Enhanced Email Validation**:

def clean\_email(self):

email = self.cleaned\_data.get('email')

pattern = r'@deakin\.edu\.au$' # Validate only Deakin emails

if not re.match(pattern, email):

raise ValidationError(\_("Email must match your Deakin email."))

return email

1. **Strong Password Validation**: Added a regex-based policy for password strength:

def clean\_password1(self):

password = self.cleaned\_data.get('password1')

pattern = r'^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$'

if not re.match(pattern, password):

raise ValidationError(\_("Password must include uppercase, lowercase, numbers, and special characters."))

## 3. Changes to urls.py

**Location**: home/urls.py

**Old Code:**

* The signup URL was missing.

**New Code:**

1. **Added Signup Path**:

path('accounts/register/', views.signup, name='signup'),

## 4. Changes to views.py

**Location**: home/views.py

**Old Code:**

* The register function lacked error handling.
* Passwords were not hashed securely.

**New Code:**

1. **Improved register View**:
   * Enhanced error handling and feedback.

def register(request):

try:

if request.method == 'POST':

form = RegistrationForm(request.POST)

if form.is\_valid():

user = form.save(commit=False)

user.set\_password(form.cleaned\_data['password1']) # Hash password securely

user.save()

otp = random.randint(100000, 999999) # Generate OTP

send\_mail(

"Verify Your Account",

f"Your OTP is: {otp}",

"no-reply@hardhat.com",

[form.cleaned\_data.get('email')],

fail\_silently=False,

)

messages.success(request, "Account created! Please verify your email.")

return render(request, 'accounts/verify\_token.html', {'otp': otp})

else:

messages.error(request, "Please correct the errors below.")

else:

form = RegistrationForm()

return render(request, 'accounts/sign-up.html', {'form': form})

except Exception as e:

messages.error(request, "An unexpected error occurred. Please try again.")

return render(request, 'accounts/sign-up.html', {'form': RegistrationForm()})

## Code for Checking the Database in Django Shell

To verify the database contents, including user data and password hashes, follow these steps:

1. Open the Django shell:

python manage.py shell

1. Import the User model:

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

1. Retrieve and display all users along with their hashed passwords:

for user in User.objects.all():

print(f"Email: {user.email}, Password Hash: {user.password}")

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

# To list all users

users = User.objects.all()

print(users)

# To display user details in a readable format

for user in users:

print(f"ID: {user.id}, Email: {user.email}, Name: {user.first\_name} {user.last\_name}")

1. Exit the Django shell:

exit()