JUNAID GIRKAR

60004190057

TE COMPS A-4

EXPERIMENT - 12

**Github Link**: <https://github.com/junaidgirkar/Django-Authentication>

**THEORY**:

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

**Complete**

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

**Versatile**

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is built with Django!

Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

**Secure**

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see Website security for more details of such attacks).

**Scalable**

Django uses a component-based “shared-nothing” architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

**Maintainable**

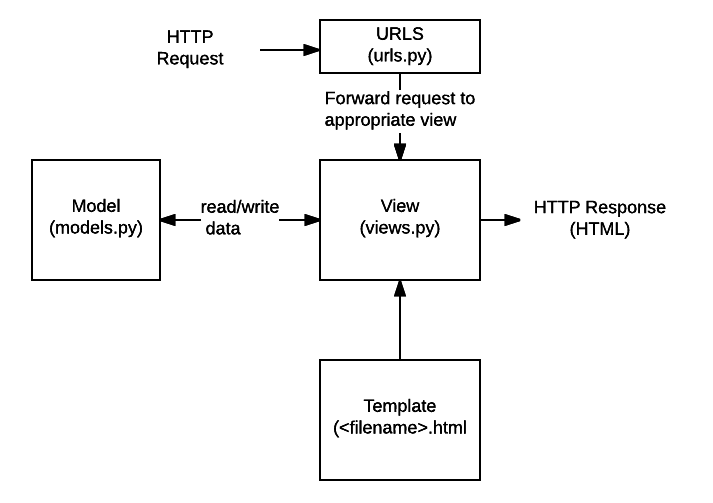
Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

**Portable**

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

in a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:



* **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
* **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
* **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
* **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

Just a few of the other things provided by Django include:

* **Forms**: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
* **User authentication and permissions**: Django includes a robust user authentication and permission system that has been built with security in mind.
* **Caching**: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
* **Administration site**: The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
* **Serialising data**: Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service (a website that purely serves data to be consumed by other applications or sites, and doesn't display anything itself), or when creating a website in which the client-side code handles all the rendering of data.

Implementing User Authentication in a Django Project:

CODE:

models.py

| from django.db import models from django.contrib.auth.models import AbstractBaseUser, BaseUserManager from django.core.validators import RegexValidator  class UserManager(BaseUserManager):  def create\_user(self, email, first\_name, last\_name, sap\_id, password=None, \*\*kwargs):  if not email:  raise ValueError("Users must have an email address")  user = self.model(  email=self.normalize\_email(email),  first\_name=first\_name,   last\_name=last\_name,  sap\_id=sap\_id,  \*\*kwargs  )  user.set\_password(password)  user.save(using=self.\_db)  return user   def create\_superuser(self, email, first\_name, last\_name, password):  user = self.create\_user(email, first\_name, last\_name, password)  user.staff = True  user.admin = True  user.save(using=self.\_db)  return user  sap\_regex = RegexValidator(regex=r"^\+?6?\d{10,12}$", message="SAP ID must be valid")  class User(AbstractBaseUser):  email = models.EmailField(max\_length=255, unique=True, default="")  first\_name = models.CharField(max\_length=64, default="")  last\_name = models.CharField(max\_length=64, default="")  sap\_id = models.CharField(validators=[sap\_regex],max\_length=12, default="")    active = models.BooleanField(default=True)  staff = models.BooleanField(default=False)  admin = models.BooleanField(default=False)   objects = UserManager()  USERNAME\_FIELD = 'email'  REQUIRED\_FIELDS = ['first\_name', 'last\_name']    def \_\_str\_\_(self):  return self.email  def has\_perm(self, perm, obj=None):  return True  def has\_module\_perms(self, app\_label):  return True  @property  def is\_staff(self):  return self.staff  @property  def is\_admin(self):  return self.admin  @property  def is\_active(self):  return self.active |
| --- |

views.py

| from django.shortcuts import render, redirect from django.contrib import messages from django.contrib.auth import authenticate, login from django.contrib.auth.decorators import login\_required from django.contrib.auth.forms import AuthenticationForm from .forms import CustomUserCreationForm, LoginForm from django.core.mail import send\_mail from django.core.mail import EmailMultiAlternatives from django.template.loader import get\_template from django.template import Context from django.core.exceptions import ValidationError from django.core.validators import validate\_email  #################### index####################################### def index(request):  try:  user = request.user  except:  user = "AnonymousUser"  return render(request, 'index.html', {'title':'index', 'user':user})  ########### register here ##################################### def register(request):  if request.method == 'POST':  form = CustomUserCreationForm(request.POST)  if form.is\_valid():  form.save()  # username = form.cleaned\_data.get('username')  email = form.cleaned\_data.get('email')  ######################### mail system ####################################  htmly = get\_template('Email.html')  d = { 'email': email }  subject, from\_email, to = 'welcome', 'your\_email@gmail.com', email  html\_content = htmly.render(d)  msg = EmailMultiAlternatives(subject, html\_content, from\_email, [to])  msg.attach\_alternative(html\_content, "text/html")  # msg.send()  ##################################################################  messages.success(request, f'Your account has been created ! You are now able to log in')  return redirect('login')  else:  form = CustomUserCreationForm()  return render(request, 'register.html', {'form': form, 'title':'reqister here'})  ################ login forms################################################### def Login(request):  if request.method == 'POST':   email = request.POST.get('email')  try:  validate\_email(email)  except ValidationError as e:  print("bad email, details:", e)  else:  print("good email")  password = request.POST.get('password')  user = authenticate(request, email = email, password = password)  if user is not None:  form = login(request, user)  messages.success(request, f' wecome {email} !!')  return redirect('index')  else:  messages.info(request, f'account done not exit plz sign in')  else:  form = LoginForm()  return render(request, 'login.html', {'form':form, 'title':'log in'}) |
| --- |

urls.py

| from .views import Login, register, index  from django.urls import include, path from django.contrib.auth import login, logout from django.contrib.auth import views as auth\_views   urlpatterns = [  path('', index,name='index'),   path('signup/', register, name='register'),  path('login/', Login, name='login'),  path('logout/', auth\_views.LogoutView.as\_view(),{'next\_page': 'index'}, name='logout'),  ] |
| --- |

forms.py

| from django.contrib.auth.forms import UserCreationForm, UserChangeForm from django.forms import ModelForm  from .models import User   class CustomUserCreationForm(UserCreationForm):   class Meta:  model = User  fields = ('email', 'first\_name', 'last\_name', 'sap\_id')   class CustomUserChangeForm(UserChangeForm):   class Meta:  model = User  fields = ('email', 'first\_name', 'last\_name', 'sap\_id')  class LoginForm(ModelForm):  class Meta:  model = User  fields = ('email', 'password') |
| --- |

setting.py

| . . .  # Application definition  INSTALLED\_APPS = [  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  'authentication',  'crispy\_forms' ] . . . |
| --- |

MyProject/urls.py

| from django.contrib import admin from django.urls import path,include  urlpatterns = [  path('admin/', admin.site.urls),  path("", include("authentication.urls")), ] |
| --- |

index.html

| {% load static %} {% load crispy\_forms\_tags %} <!DOCTYPE html> <html lang="en">  <head>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <meta name="title" content="project">  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  <meta name="language" content="English">   <title>{{title}}</title>    <!-- bootstrap file -->  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" />  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  <!-- bootstrap file-->   <!-- jQuery -->  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">    <!-- main css -->  <link rel="stylesheet" type="text/css" href='{% static "index.css" %}' />    <!-- message here -->   {% if messages %}  {% for message in messages %}   <script>  alert("{{ message }}");  </script>   {% endfor %}  {% endif %}   <!--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-->    </head>  <body class="container-fluid">    <header class="row">   <!-- navbar-->  <nav class="navbar navbar-inverse navbar-fixed-top">  <div class="container-fluid">  <div class="navbar-header">  <button class="navbar-toggle" data-toggle="collapse" data-target="#mainNavBar">  <span class="icon-bar"></span>  <span class="icon-bar"></span>  <span class="icon-bar"></span>  <span class="icon-bar"></span>  </button>  <a class="navbar-brand" class="styleheader" href='{% url "index" %}'>project</a>  </div>  <div class="collapse navbar-collapse" id="mainNavBar">  <ul class="nav navbar-nav navbar-right">  <li><a href="{% url 'index' %}">Home</a></li>   {% if user.is\_authenticated %}  <li><a href="{% url 'logout' %}"><span class="glyphicon glyphicon-log-out"></span> &nbsp; Logout</a></li>  {% else %}  <li><a href="{% url 'register' %}"><span class="glyphicon glyphicon-user"></span> &nbsp; Sign up</a></li>  <li><a href="{% url 'login' %}"><span class="glyphicon glyphicon-log-in"></span> &nbsp; Log in</a></li>  {% endif %}   </ul>  </div>  </div>  </nav>  </header>  <br/>  <br>  <br>  <div class="row">  {% block start %}  {% if user.is\_authenticated %}  <center><h1>welcome back {{user.email}}!</h1></center>  {% else %}  <center><h1>log in, plz . . .</h1></center>  {% endif %}  {% endblock %}  </div> </body>  </html> |
| --- |

login.html

| {% extends "index.html" %} {% load crispy\_forms\_tags %} {% block start %}   <div class="content-section col-md-8 col-md-offset-2">  <center>  <form method="POST" style="border: 1px solid black; margin: 4%; padding:10%; border-radius:1%;">  {% csrf\_token %}  <fieldset class="form-group">  <label for="email">Email</label><br>  <input id="email" type="text" autocomplete="off" name="email" value="" class="form-control input-lg" placeholder="Email" />  <br><br>  <label for="Password">Password</label><br>  <input id="password" type="password" autocomplete="off" name="password" value="" class="form-control input-lg" placeholder="Password" />   </fieldset>  <center>  <button style="background: black; font-size: 2rem; padding:1%;" class="btn btn-outline-info" type="submit"><span class="glyphicon glyphicon-log-in"></span> &nbsp; login</button>  </center>  <br/>  <sub style="text-align: left;"><a href="{% url 'register' %}" style="text-decoration: none; color: blue; padding:2%; cursor:pointer; margin-right:2%;">don't have account,sign up</a></sub>  </form> </center>  </div>  {% endblock start %} |
| --- |

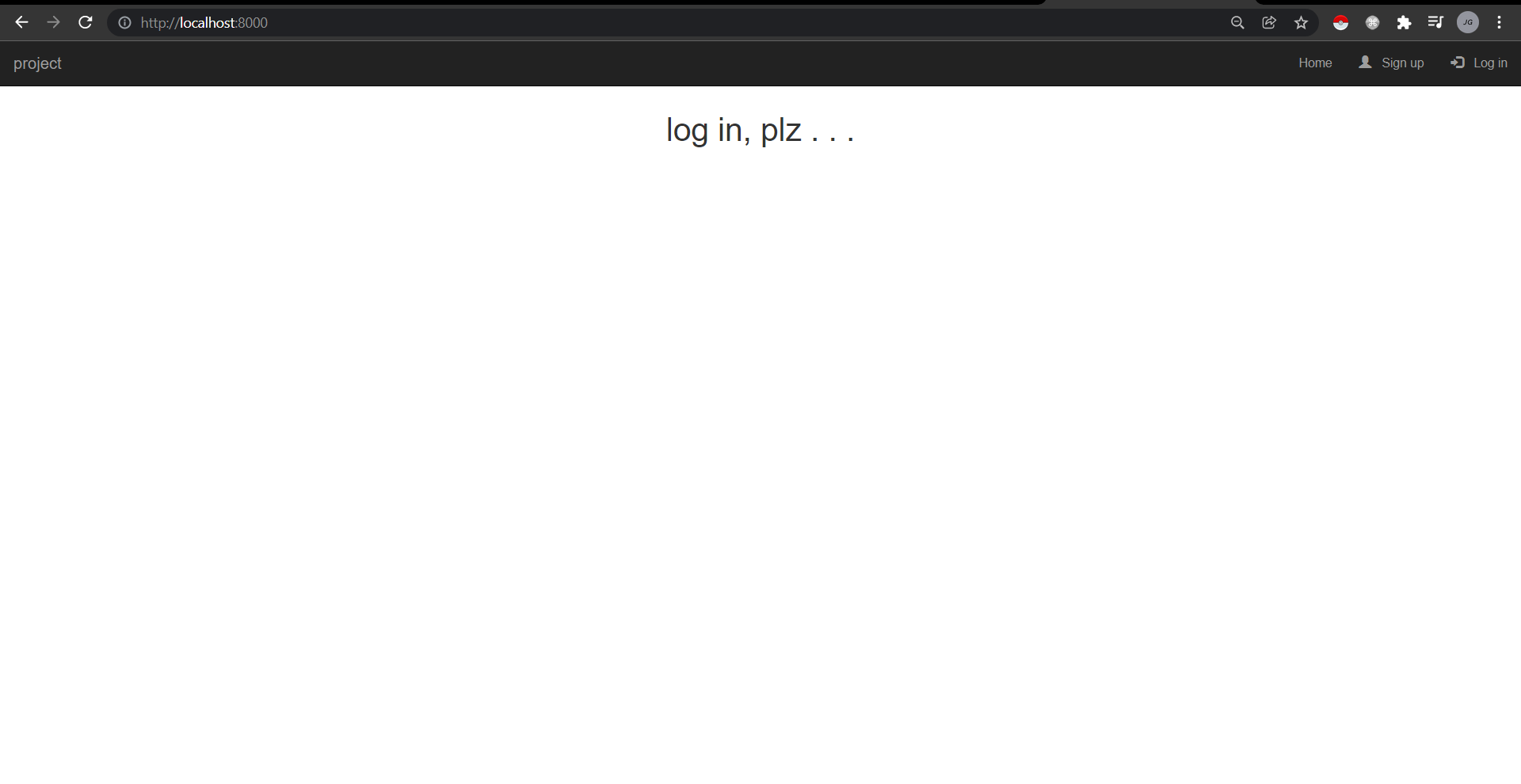
register.html

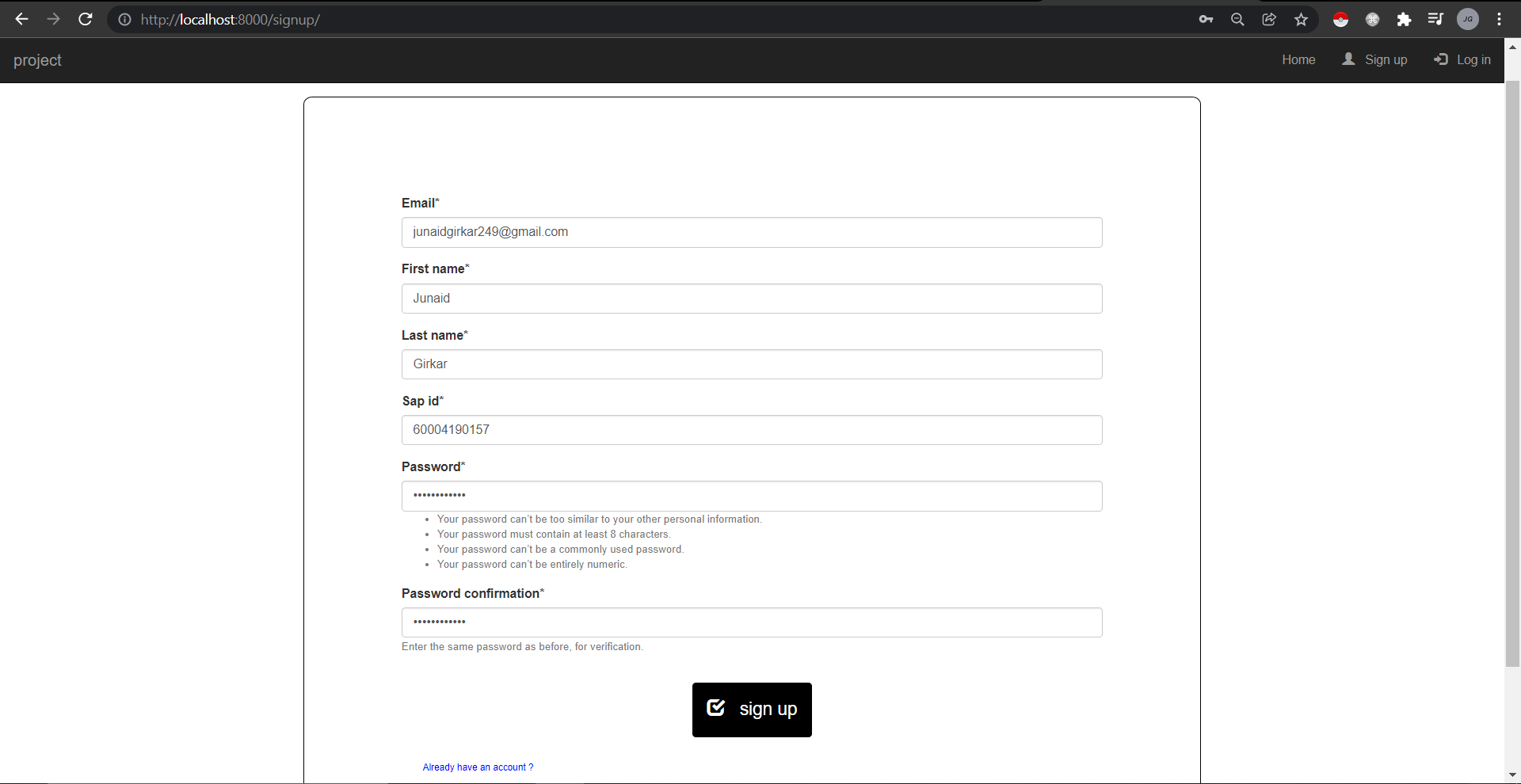
| {% extends "index.html" %} {% load crispy\_forms\_tags %} {% block start %}  <div class="content-section col-md-8 col-md-offset-2">  <form method="POST" style="border: 1px solid black; margin: 4%; padding:10%; border-radius:1%;">  {% csrf\_token %}  <fieldset class="form-group">  {{ form|crispy}}  </fieldset>  <center>  <button style="background: black; padding:2%; font-size: 2rem; color:white;" class="btn btn-outline-info" type="submit"><span class="glyphicon glyphicon-check"></span> &nbsp; sign up</button>  </center>  <br />  <sub><a href="{% url "login" %}" style="text-decoration: none; color: blue; padding:3%; cursor:pointer;">Already have an account ?</a></sub>  </form> </div> {% endblock start %} |
| --- |

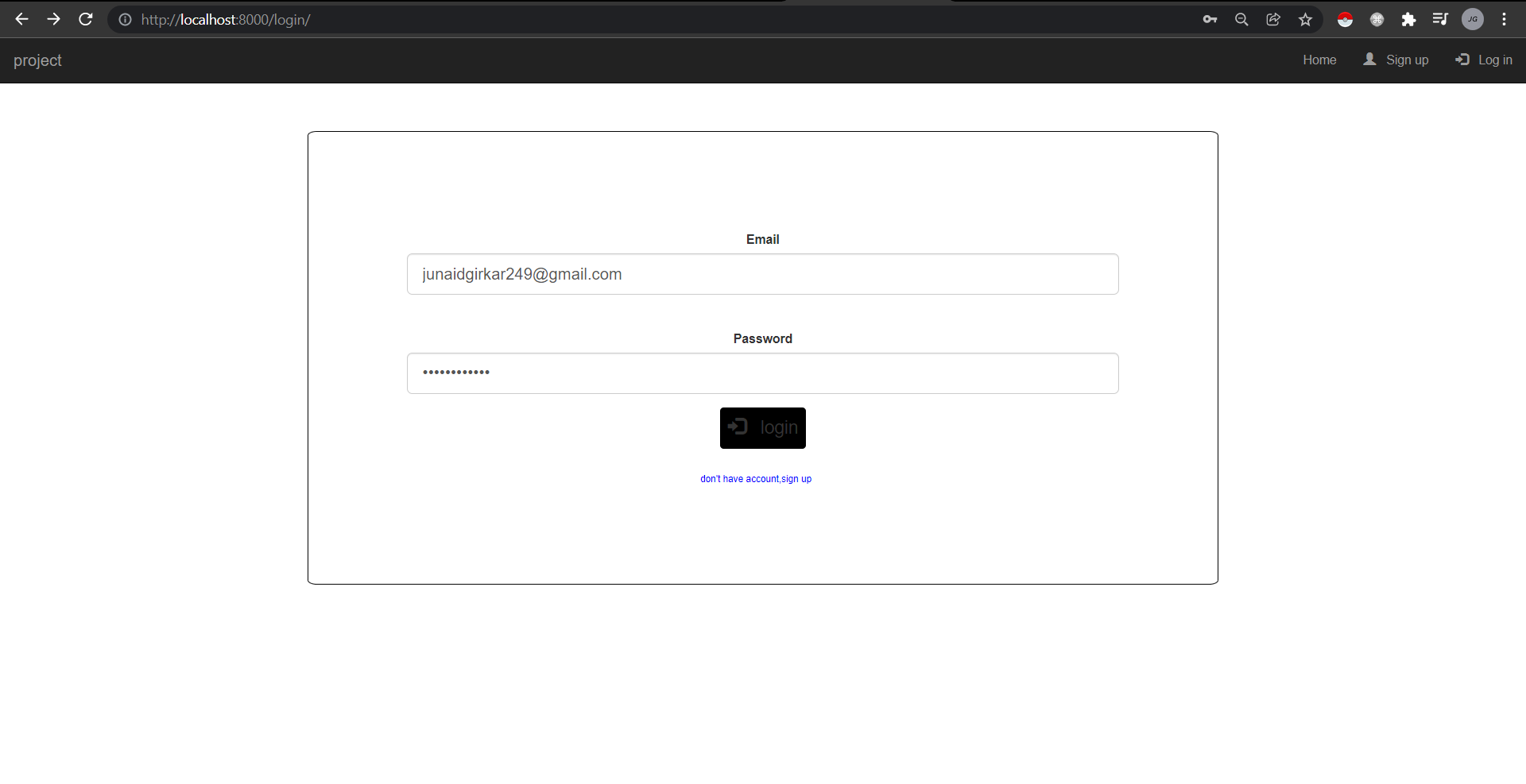
email.html

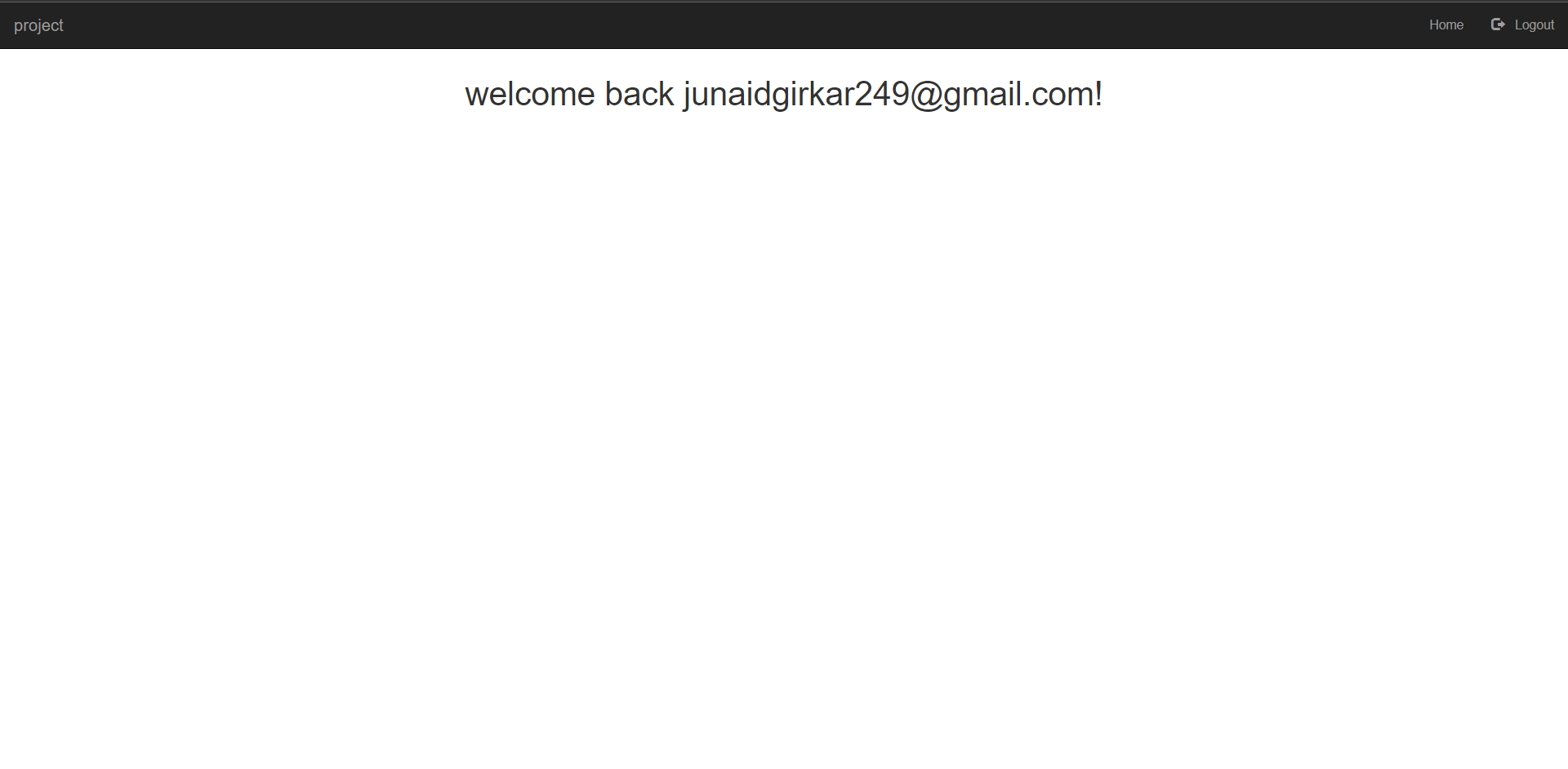
| <!DOCTYPE html> <html lang="en" dir="ltr">  <head>  <meta charset="utf-8">  <title></title>  <style>  @import url('https://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900');  </style>  </head>  <body style="background: #f5f8fa;font-family: 'Roboto', sans-serif;">  <div style="width: 90%;max-width:600px;margin: 20px auto;background: #ffffff;">  <section style="margin: 0 15px;color:#425b76;">  <h2 style="margin: 40px 0 27px 0;text-align: center;">Thank you to registration</h2>  <hr style="border:0;border-top: 1px solid rgba(66,91,118,0.3);max-width: 50%">  <p style="font-size:15.5px;font-weight: bold;margin:40px 20px 15px 20px;">Hi {{username}}, we have received your details and will process it soon.</p>  </section>  </div>  </body> </html> |
| --- |

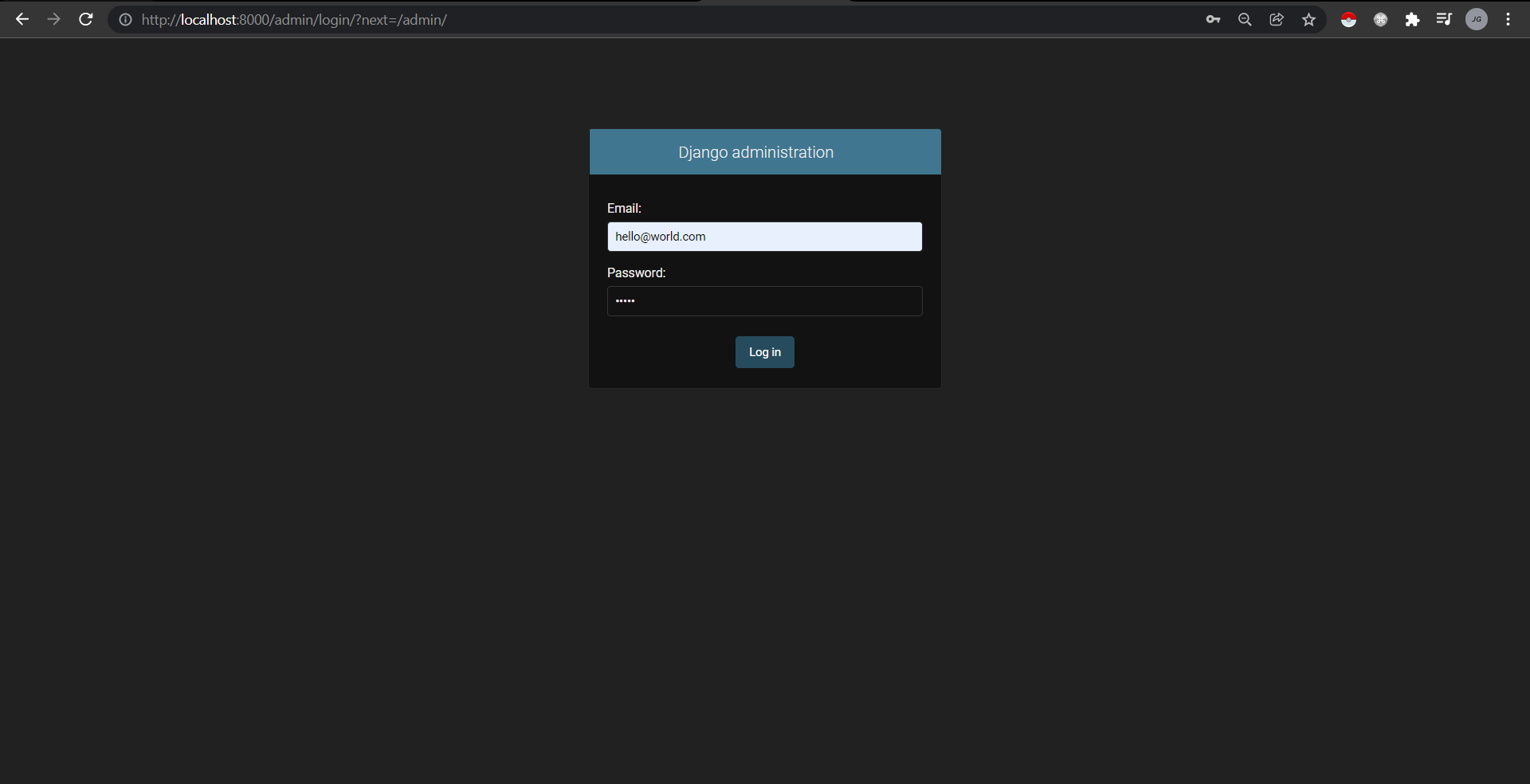
OUTPUT:

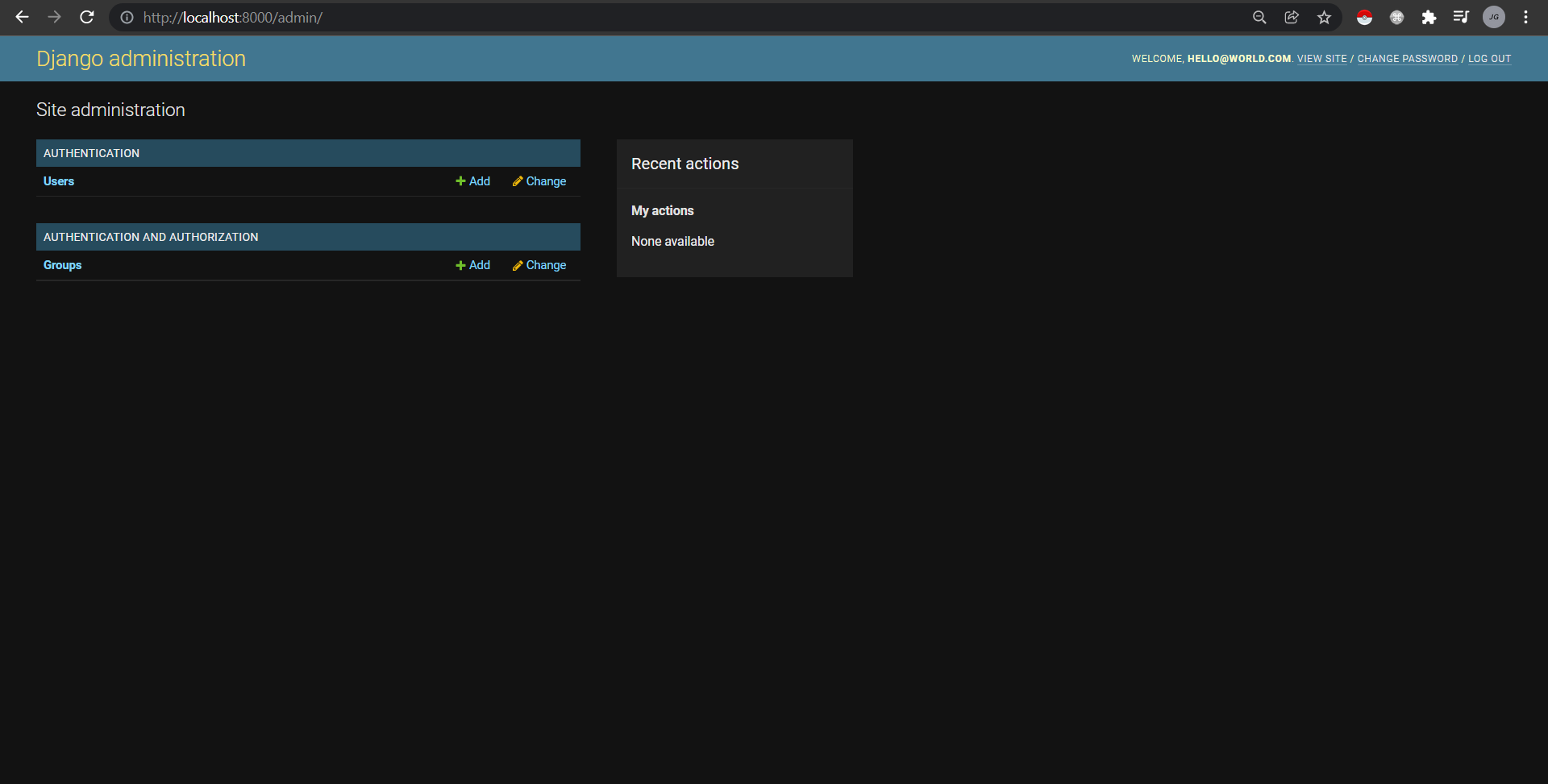
Home Page:

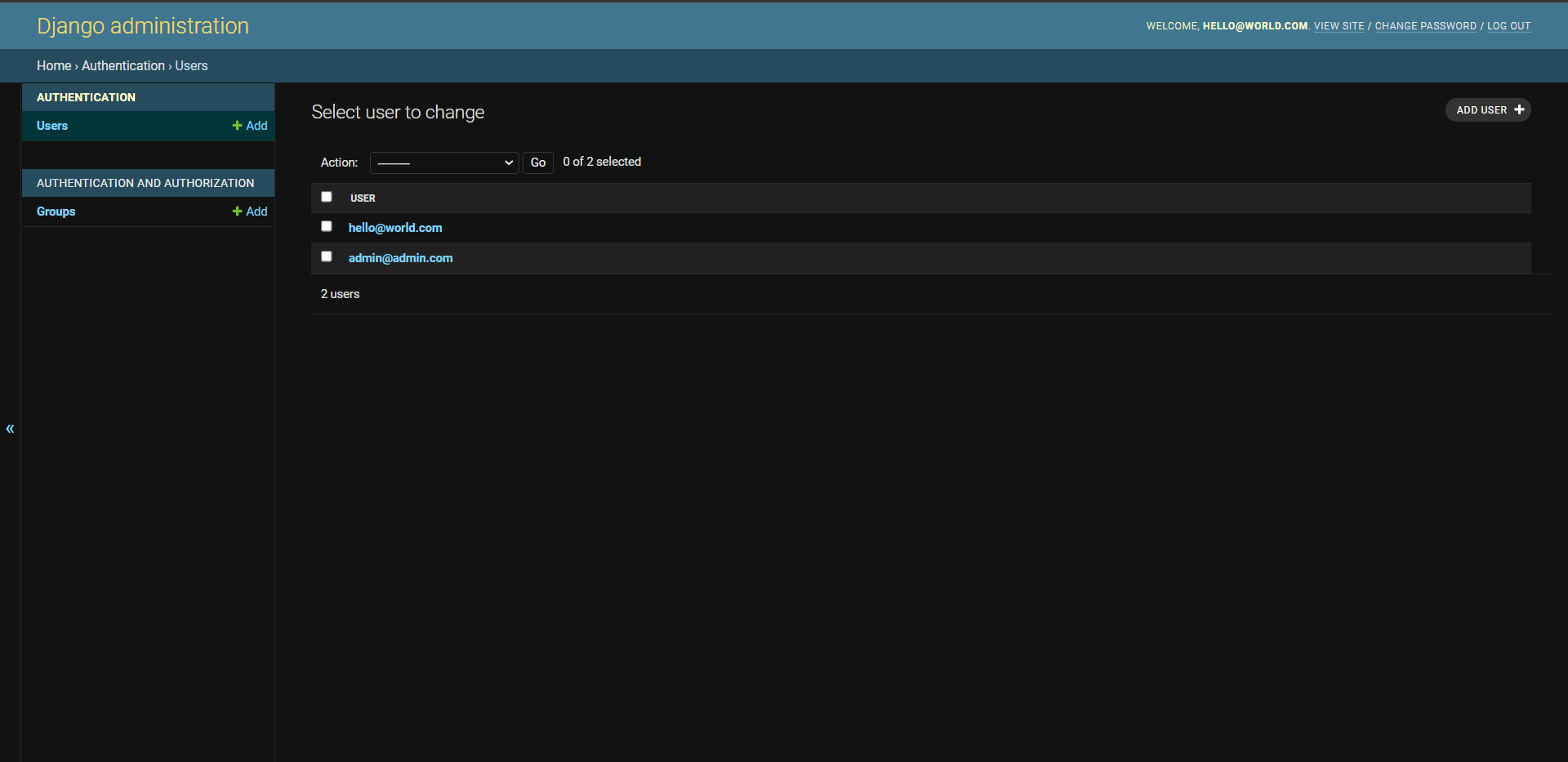
Register Page:

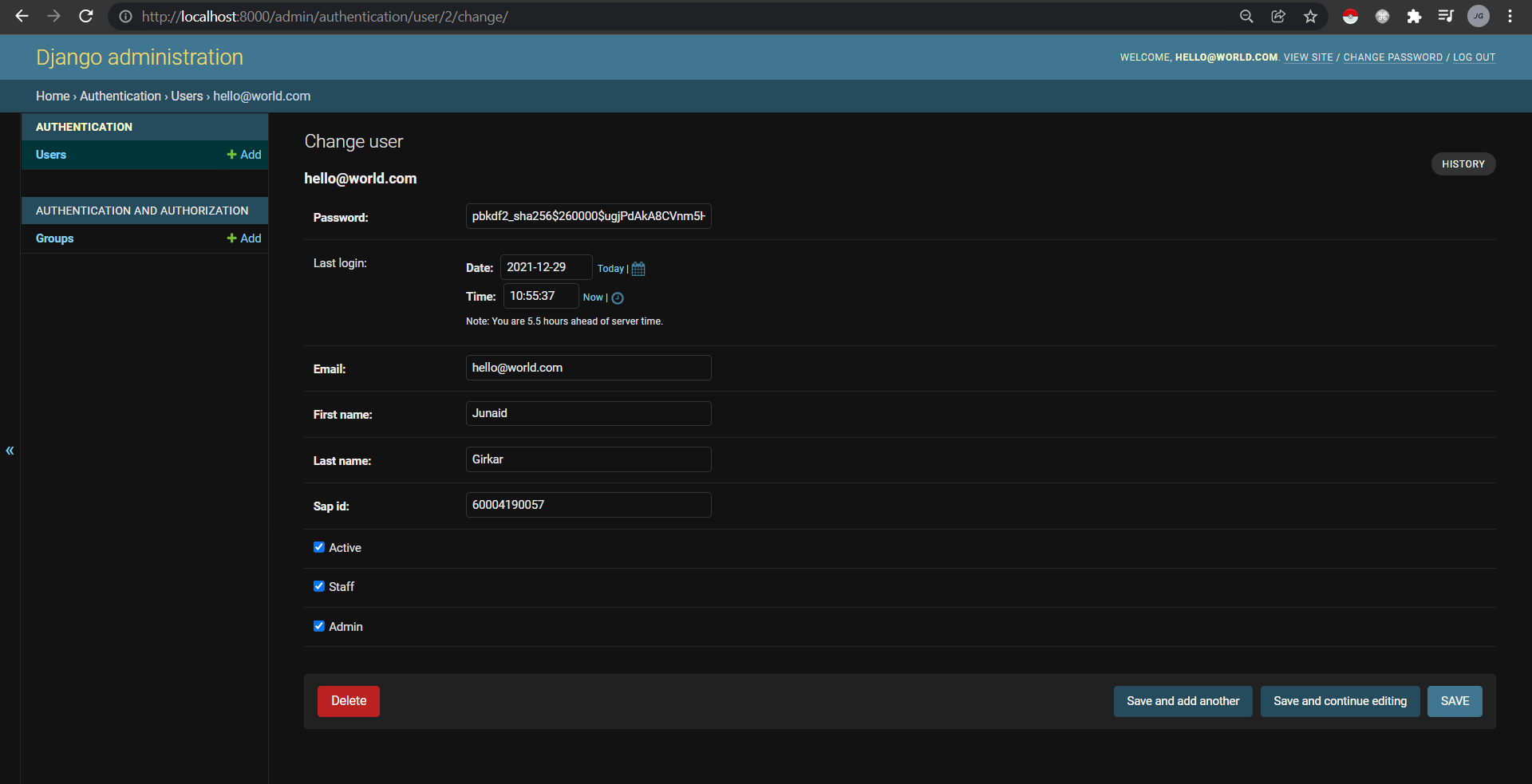
Login Page:

Home Page (After successful login):

Admin Panel Login:

Admin Home Page:

Users List Page in Admin

User Detail Page in Admin

**Conclusion:**

We learnt about the Django web development framework and its functionalities. We then created a website with a user authentication system with a custom Abstract Base User model.