**DAY – 1**

**18-03-2024**

**Objectives:**

**->** Django introduction

-> Exploring First project creation in Visual Studio Code using Django

-> URL and dynamic URL creation

**Django Introduction:**

What is framework?

A framework is like a structure that provides a base for the application development process.

- **Django** is one of the python frame which is used to create a dynamic web

application.

- **Architectures:** MVC-Model View Controller, MVT - Model view and template

- **Adrian Holovaty** and **Simon Willison** introduced in 2003 and officially in 2005

- **ADV:** ORM-Query in Django

Administration GUI

Development Environment

Less Coding

Don’t Repeat Yourself (DRY)-Using Function (Reusability)

Fast Development

Applications in which it is used: Instagram, Mozilla, Pinterest, bitbucket, The Washing Ton time (American newspaper).

The **main databases** that Django works on are Oracle, MySQL, PostgreSQL, and SQLite.

**Feature:**

- Scalability - Can flexible to any OS

- Security - Authentication system

- Open Source - As one can use

- Vast and Support Community -

- Rapid Development -

**MVC:** Data base files can be stored (Front end part will be stored)

**MVT:** Django this pattern (Front end comes under templates, Model- can store database connections, Controller-View is used to control the template and model.

Request->URL File-> View (Used to move based on the users request we should move either to frontend or backend)

**Sample Project Creation:**

-> Proper Location – Desktop (Folder Name: Django\_AIML\_ADITYA)

-> Open the command prompt in that folder location.

-> For the project creation Django provides one command

**syntax for starting a project** - django-admin startproject <proj\_name>

**Ex:**

django-admin startproject FirstProject

-> That command in command Prompt will create a file in the the folder-Django\_AIML\_ADITYA.

In **FirstProject File: (In VSCode)**

-> **init.py:** This is ana empty Python file.

-> **asgi.py:** asynchronous server gateway interface.

-> **wsgi.py:** Web server gate way interface.

-> **settings.py:** We can see all project settings, Every settings file contains one authentication secret

key.

-> **urls.py:** admin is the default URL in Django, here we can create the user-defined urls

-> **Server file:** manage.py is the Django serve file (python file)

-> **manage.py**- we can’t run server in Django.

-> **MIDDLEWARE**

-> **Templates**-Backend code for running the html files

-> **database -** db.sqlite

-> AUTH-PASSWORD­­­­­­­\_VALIDATORS

-> Static Files

-> To create an application, we change the application to the FirstProject folder,

-> For app creation django provides another command

**Application creation syntax in django:** python manage.py startapp appname

**Ex:** python manage.py startapp FirstApp

One more folder is created with the name of FirstApp in the Django\_AIML\_Aditya-

>FirstProject and even another folder in the vscode.

->**FirstApp: (In vscode)**

**init.py:** it is an empty python file.

**admin.py:** Django administration page we use this, even to see the administration

GUI information and we can add data related configurations.

**apps.py:** It contains django application information.  
 **models.py:** to store the database file or information.  
 **views.py:** It is a controller file in django and it controls both templates and models.  
 **test.py:** It used to test the testcase in django.

**->How to run django server?**

**Command:** python manage.py runserver

**O/P:** Some statements+ Django Package Version + Month + Date + Time +Django

Default code server.

To see django in the browser to check whether the server is in working will be seen by **copying the default code server and paste in the browser link** and it navigate to a place where it shows the **installed worked server successfully.(This indicates the server is in working condition).**

**Ctrl+c** is used to break the run the server.

**-> How to create Url in django?**

If we want to send any request in browser we sent by using url so we need it.

**Creating user defined url:**

To create the url we use the path function and parameters: **path(‘urlname/’, views.function =name, name of the url)**

In django all the files should be created in views. Here server in the sense cmd. We get the error so import in the the views from the app name.

From FirstApp import views

We get attribute error:

We write some instructions in views.py in FirstApp:

From django.http import HttpResponse

Def he(request):

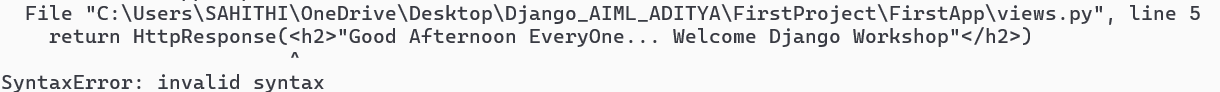
Return HttpResponse(“Good Afternoon EveryOne.. Welcome to the django

workshop”)

In browser we get page not found error so we need to extend the url by / hello/ we get the response of the text.

Even can use the heading tags for display the test effectively.

If we are having any errors in the code it is immediately given in the server with the errors and its respective line number.



To increase the font size we use:

 return HttpResponse("<h2>Good Afternoon EveryOne... Welcome Django Workshop</h2>")

Without errors the server will be like:





Put a comma after a path instruction other wise we get errors

To create the new url in urls.py file in FirstProject:

path('sample/',views.sam,name='Sample')

We get an error:

So in views.py file we create a function sam:

def sam(request):

    return HttpResponse("<center>Hello World</center>")

Even we can use the style tags, center tags in the HttpResponse:

def sam(request):

    return HttpResponse("<h2 style = color:red><center>Hello World</center></h2>")

Appling Background-color, font-color, font-size so we use:

def sam(request):

    return HttpResponse("<h2 style = color:red;background-color:lightblue;font-size:45px;font-style:italic><center>Hello World</center></h2>")

O/P:

****

**Dynamic urls creation:**

**Syntax:** Path(‘urlname\<data type:variable name>/’,view.function name, nameof the the url)

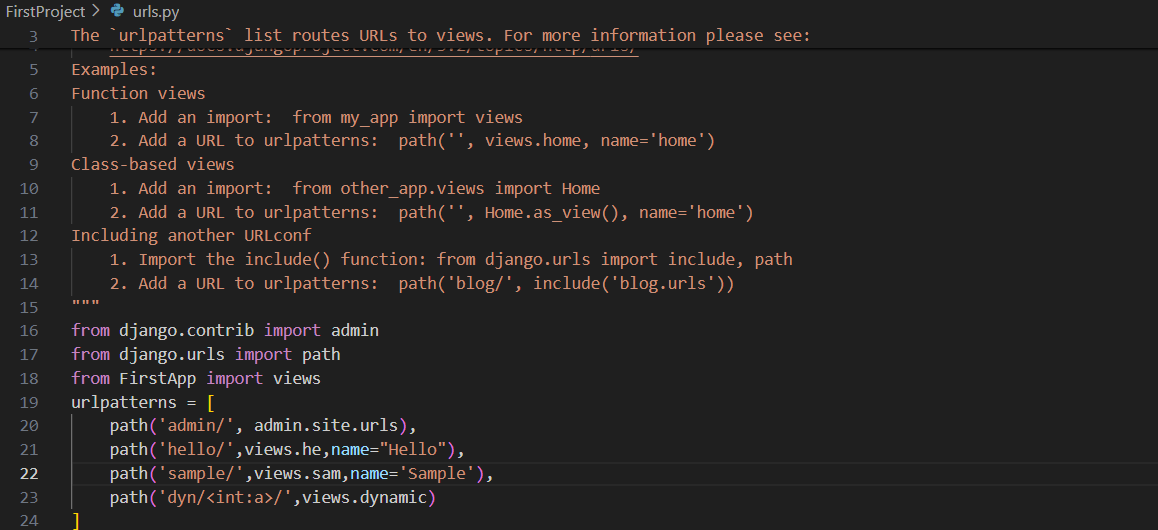
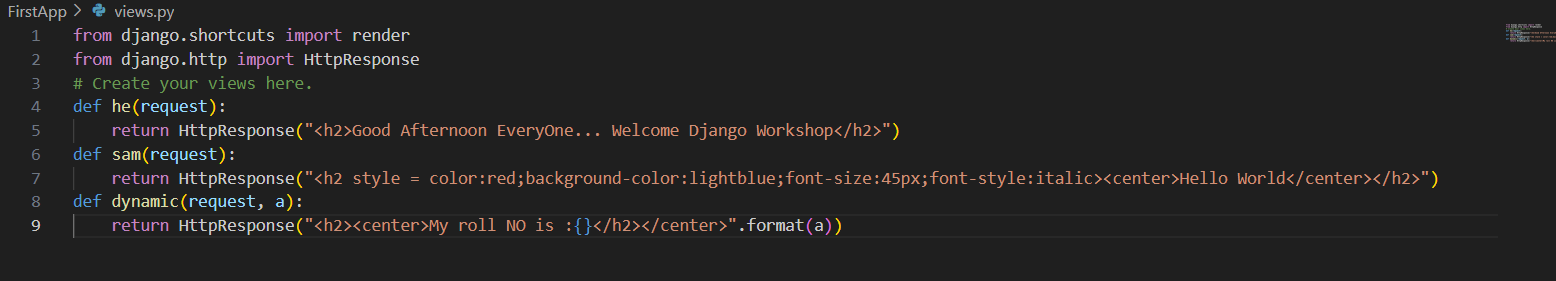
path('dyn/<int:a>/',views.dynamic)

def dynamic(request, a):

    return HttpResponse("<h2>My roll NO is :{}</h2>".format(a))



**The total program:**

**In views:**

**DAY – 2**

**19-03-2024**

**Objectives:**

-> Templates and DTL

-> Working on CSS

-> Static File Handling

In addition to it we can also add the other datatypes into the display by modifying the dynamic function as:

def dynamic(request, a, b):

    return HttpResponse("<h2><center>My roll NO is :{} and my name is {}</h2></center>".format(a,b))

save and run as: <http://127.0.0.1:8000/dyn/9/sahithi/>

**O/P:**



**Template:**  
.Html files and front-end part will be stored in these templates. We use render function:

->render (parameter-1, parameter-2, parameter-3)

render (http request, ’html filename’, dictionary)

User->Django server-> urls.py ->views.py->.html

Even we are having the inbuild importing of render in views.py.

**How to create a Templates in Django?**

->Django app folder-->First app-->New folder -->Templates.

-> In templates folder 🡪Consider a new folder and mention that file name as temp.html. Do

this in the Visual Studio code itself where it also appears in the folder also.

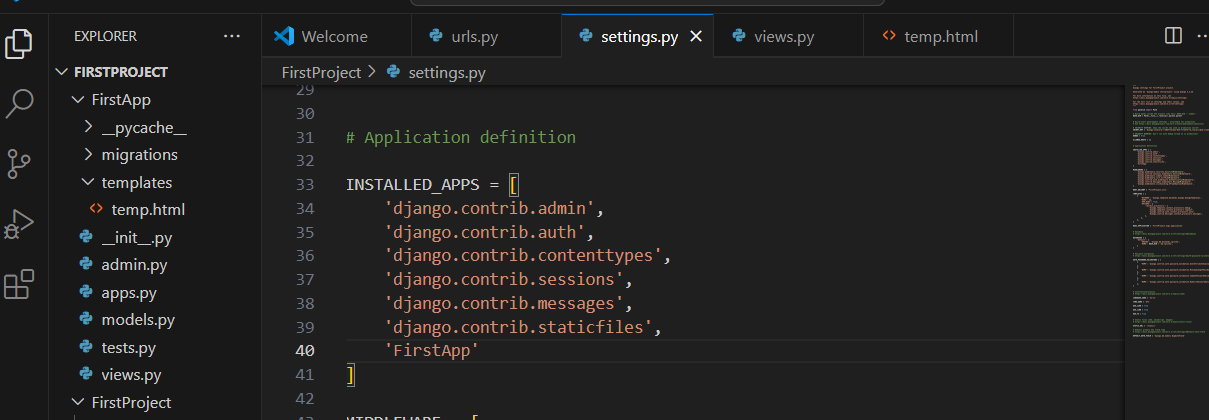
->In an editor to work with the html file we use the shortcut<html(In sublime)



<http://127.0.0.1:8000/temp/>

we get the error:

We need to register our app in settings.py so that after saving that app then it appears the output in the website.

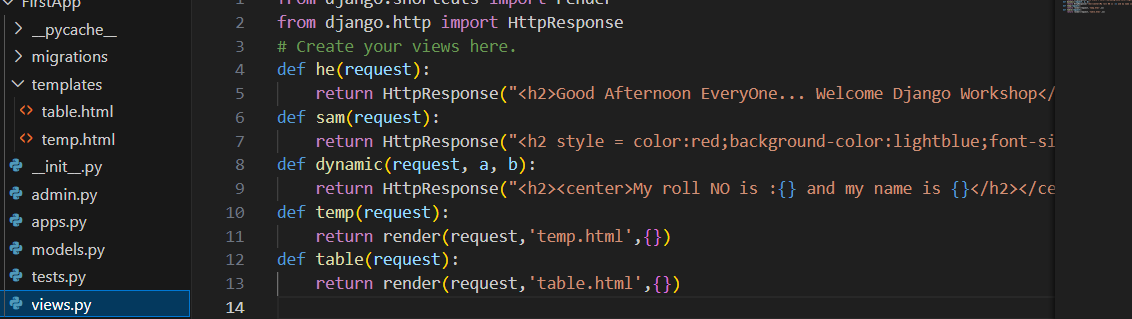


We can also style the tags by the below instruction:

            <h2 style="color:cyan;background-color: blue;font-size:40px">Welcome to django templates</h2>

We add one more path in urls.py and new function in views.py and created a new file in the templates

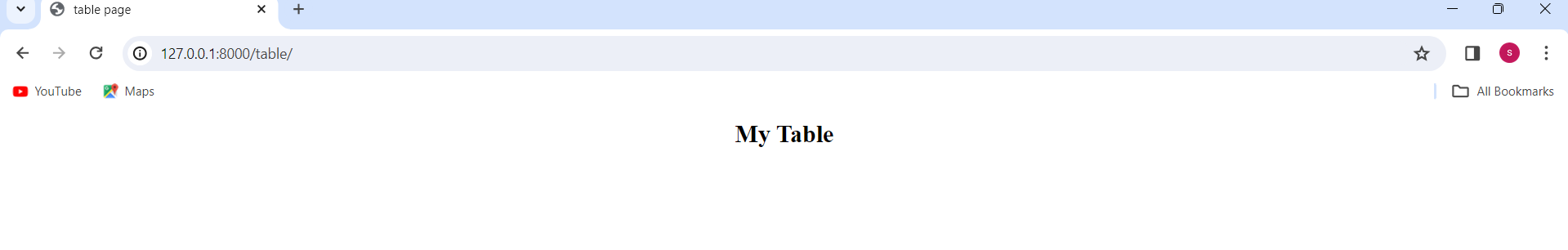


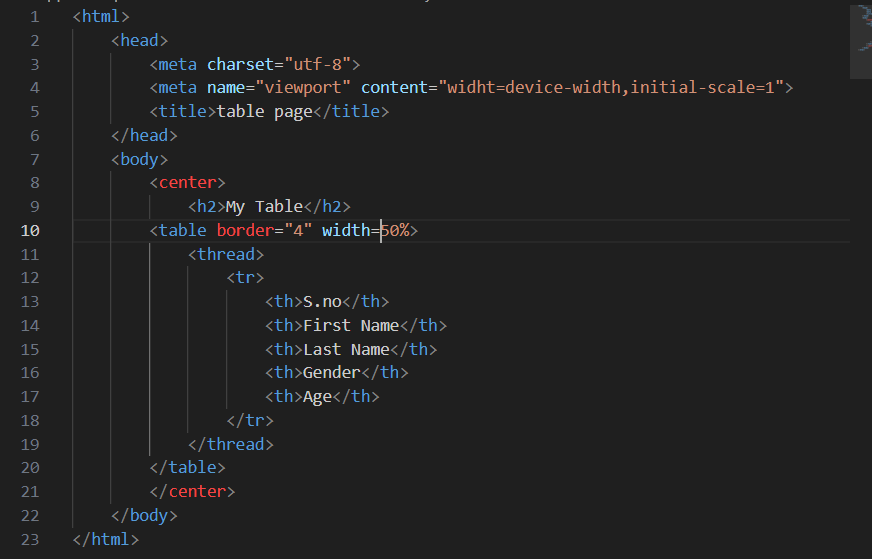


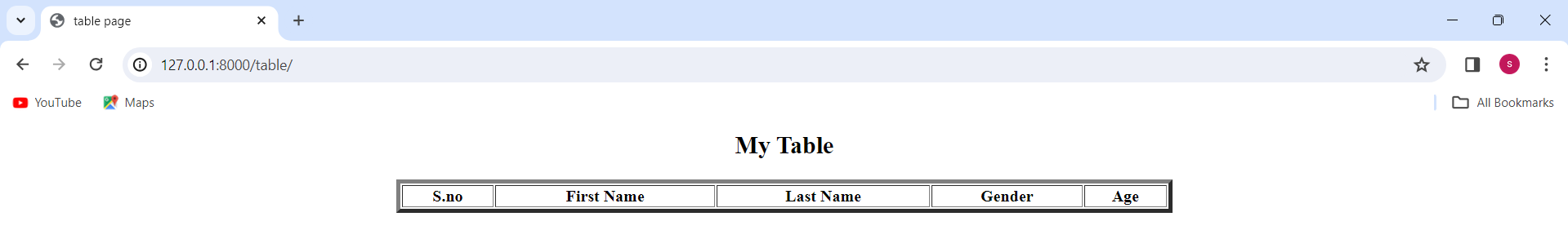
**Inside the table function:**

We:





**Now creation of table and applying the borders to it:**  




<th> - Table header

<td> - Table Data

**Code:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>table page</title>

</head>

<body>

<center>

<h2>My Table</h2>

<table border="4" width=50% bgcolor="cyan" bordercolor="black">

<thread>

<tr>

<th>S.no</th>

<th>First Name</th>

<th>Last Name</th>

<th>Gender</th>

<th>Age</th>

</tr>

</thread>

<tbody>

<tr>

<td>1</td>

<td>Sri</td>

<td>K</td>

<td>Female</td>

<td>9</td>

</tr>

<tr>

<td>2</td>

<td>Sam</td>

<td>p</td>

<td>Female</td>

<td>8</td>

</tr>

<tr>

<td>3</td>

<td>Sanjana</td>

<td>S</td>

<td>Female</td>

<td>19</td>

</tr>

<tr>

<td>4</td>

<td>Anu</td>

<td>K</td>

<td>Female</td>

<td>19</td>

</tr>

</tbody>

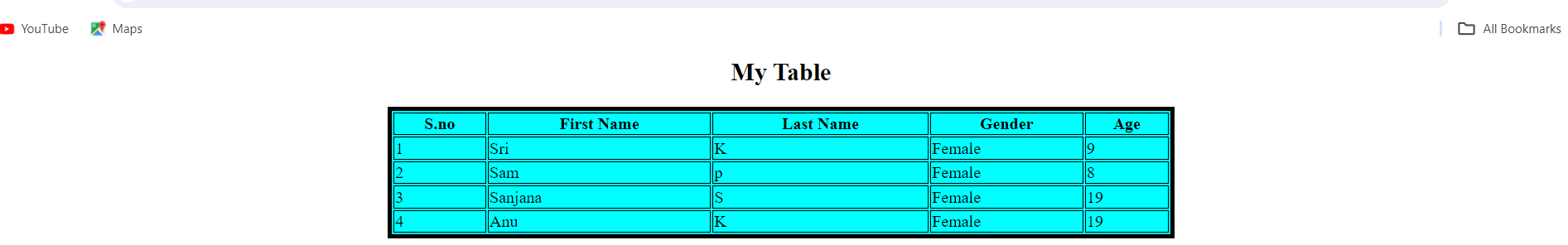
</table>

</center>

</body>

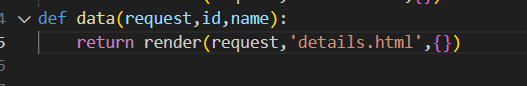
</html>

**O/P:**



**Dynamic url creation:**

**In url file:**  


**In views file:**  


**In details.html file:**  
<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>Data page</title>

</head>

<body>

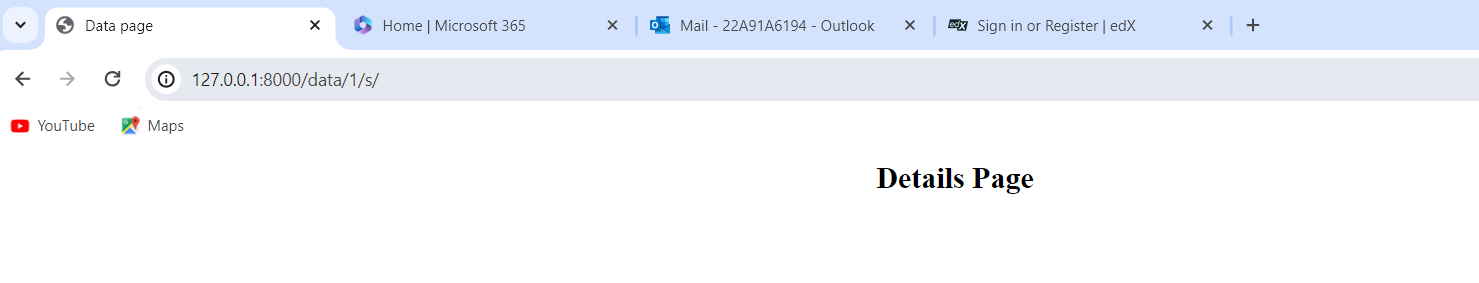
<center>

<h2>Details Page</h2>

</center>

</body>

</html>



To print any data in the html file we use **DTL-(Django Template Language).**

**Conditional Statements:**

->{% if (condition)%}

->{% endif %}

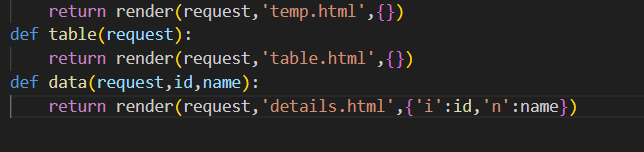
**Loops:**

{% for i in variable name %}

{{i}}

{% end for %}

**In views:**



**Code in detail.html will be:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>Data page</title>

</head>

<body>

<center>

<h2 style="color:lightgreen;font-size:40px;">Details Page:</h2>

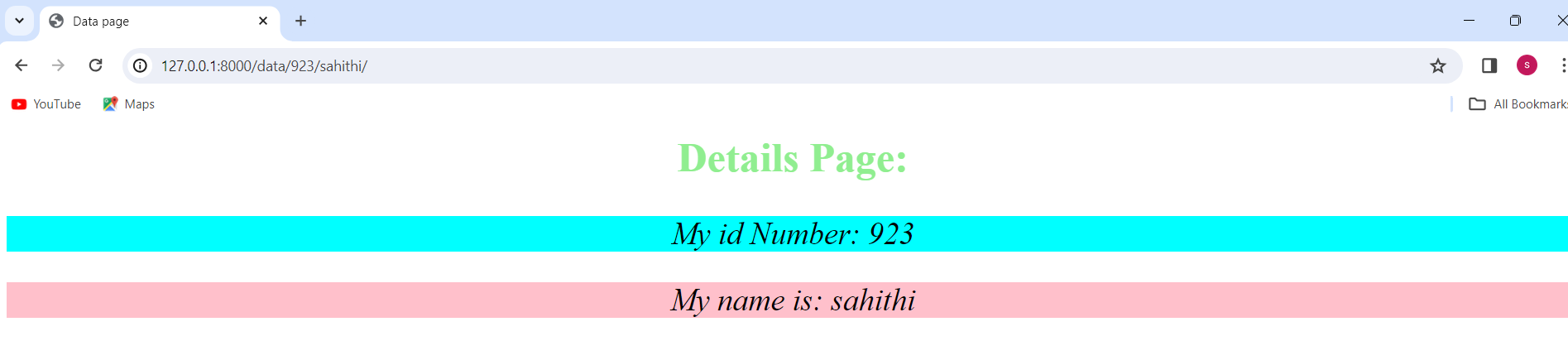
<p style="background-color: cyan; font-size:30px;font-style: italic;">My id Number: {{i}}</p>

<p style="background-color:pink;font-size:30px;font-style:italic">My name is: {{n}}</p>

</center>

</body>

</html>



**CSS:**

CSS - Cascading Style Sheets

We use css to provide the styles on the html tags.

**Syntax:**

Property:value;

**Types of CSS:**

**->Inline CSS -** to provide the css properties in a single line or same line is called as inline**.**

**->Internal CSS -** To provide styles or css properties in head tag.

**->External CSS -** We can create separate .css file in that we can apply css properties to the

html tag.

If we want to send any data from controller to template we use dictionary.

**Inline CSS:**

Now we create a **new path** :

path('inline/',views.inline,name='inline')

**and in views:**

def inline(request):

    return render(request,'inline.html',{})

**Code in inline.html:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>inline page</title>

</head>

<body>

<center>

<h2 style="text-align: center;font-size:50px;background-

color:powderblue;color:black;font-style:italic">Inline CSS</h2>

</center>

</body>

</html>



**Using the padding and margin attributes:**

**Code change will be:** <h2 style="text-align: center;font-size:50px;background-color:powderblue;color:black;font-style:italic;padding:30px;margin-left: 200px;margin-right: 200px;margin-top: 30px;">Inline CSS</h2>

****

As the whole styling is in a single line we call it an inline CSS.

**Internal CSS:**

**New path/URL:**

path('internal/',views.internal,name="internal")

**New function in views:**

def internal(request):

    return render(request,'internal.html',{})

**Creation of new html file as internal.html:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>Internal page</title>

</head>

<body>

<center>

<h2>Internal CSS</h2>

</center>

</body>

</html>

Now applying the internal css then the code changes to the below:  
**Code:**

**{H2 indicates the internal CSS}**  
<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>Internal page</title>

<style type="text/css">

h2{

text-align:center;

padding: 30px;

margin-right: 200px;

margin-left:200px;

margin-top: 30px;

background-color: aquamarine;

color:black;

font-style:italic;

font-size: 40px;

}

</style>

</head>

<body>

<center>

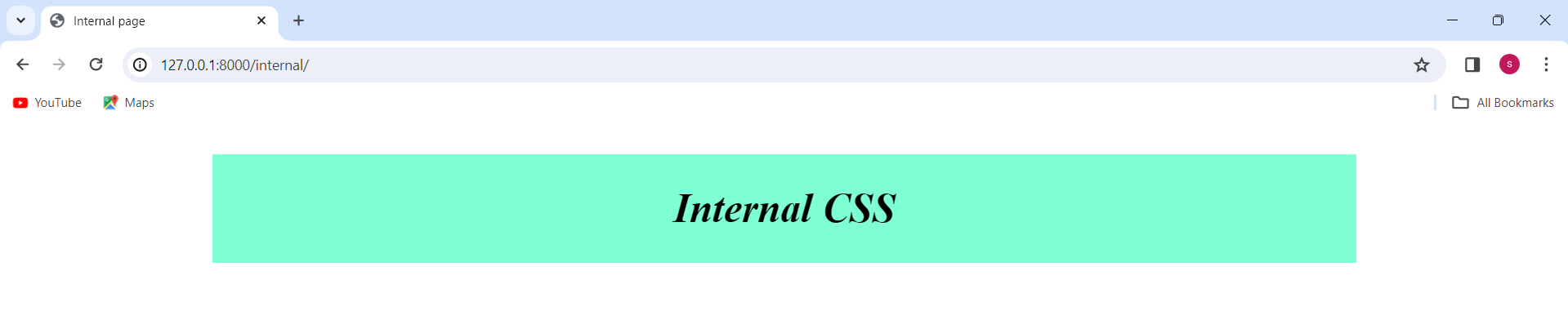
<h2>Internal CSS</h2>

</center>

</body>

</html>

**O/P:**



**TO start server:** python manage.py runserver

**To break the server:** Ctrl+C

**External CSS:**  
Create **a new url** in in url.py

path('external/',views.external,name="external")

Create a **new function** in views.py

def external(request):

    return render(request,'external.html',{})

Create a **external.html file**

<html>

    <head>

        <meta charset="utf-8">

        <meta name="viewport" content="widht=device-width,initial-scale=1">

        <title>External page</title>

    </head>

    <body>

        <h2>Regristration Form</h2>

        <form>

        </form>

    </body>

</html>

We can create one separate .css file in that we can apply css properties to the html tag

-username

-mobile no

-email

-psw

-cpsw

**Code will be:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>External page</title>

</head>

<body>

<h2>Regristration Form</h2>

<form>

Name: <input type="text" name="uname" placeholder="Enter your

Username"><br><br>

Mobile Number: <input type="number" name="mbl" placeholder="Enter your mobile

Number"><br><br>

E-mail: <input type="email" name="email" placeholder="Enter your email

address"><br><br>

password: <input type="password" name="psw" placeholder="Enter

paswword"><br><br>

conformation password: <input type="password" name="cpwd" placeholder="Enter a

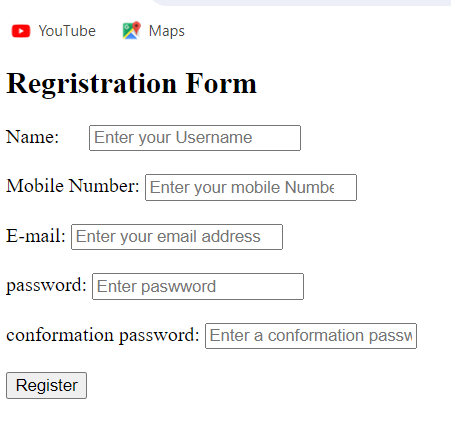
conformation password"><br><br>

<button>Register</button>

</form>

</body>

</html>

O/P:  


**Static File Handling:**  
We can store .css files, .js files, images, bootstrap links

How to create static files?

* First app->create a newFolder namely as static (folder)
* In Static folder we create a 3 subfolders

->css

->js

->images

Now **create a folder in css** as sample.css,

To provide the interconnection in between the html file and css file we use **link tag**. Where in side the **href** we are mentioning the sample.css file but yet we get an error so we need to import the file by **{% load static %}.**

**Code will be of sample.css:**

h2{

text-align: center;

color:black;

font-style: italic;

font-size:40px

}

**In html code:**

{% load static %}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>External page</title>

<link rel="stylesheet" type="text/css" href="{% static 'css/sample.css'%}"

</head>

<body>

<h2>Regristration Form</h2>

<form>

Name: <input type="text" name="uname" placeholder="Enter your

Username"><br><br>

Mobile Number: <input type="number" name="mbl" placeholder="Enter your mobile

Number"><br><br>

E-mail: <input type="email" name="email" placeholder="Enter your email

address"><br><br>

password: <input type="password" name="psw" placeholder="Enter

paswword"><br><br>

conformation password: <input type="password" name="cpwd" placeholder="Enter a

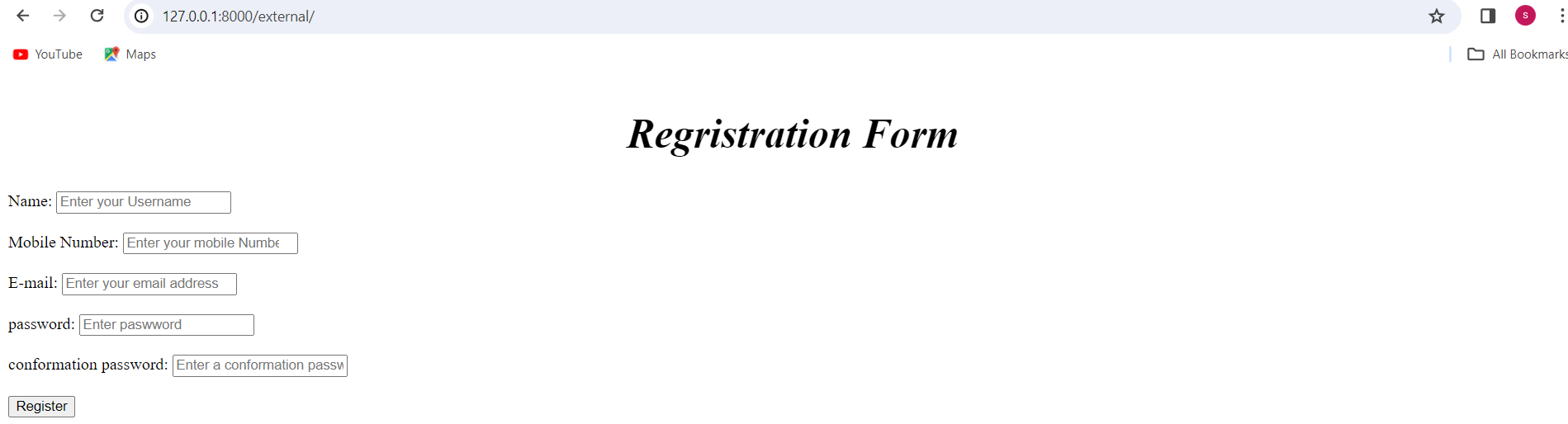
conformation password"><br><br>

<button>Register</button>

</form>

</body>

</html>

**O/P:**

Now applying **styling for form tag** in **external css** and the code for form tag will be:

form{

background-color: lavenderblush;

margin-left: 30%;

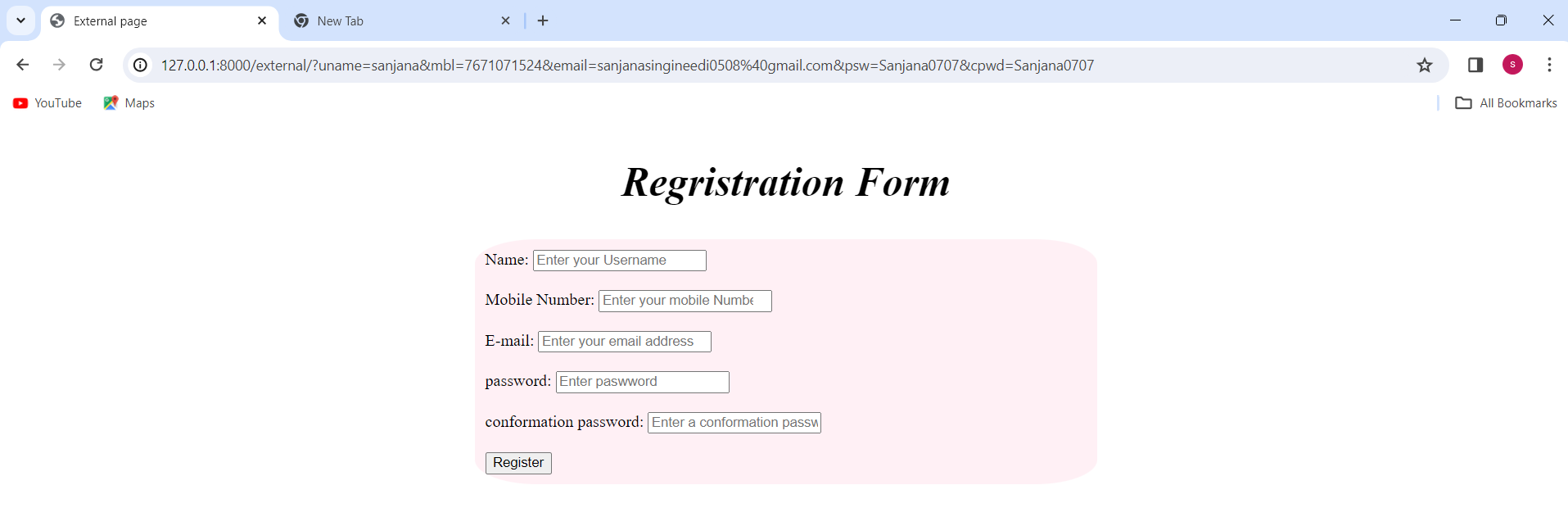
margin-right: 30%;

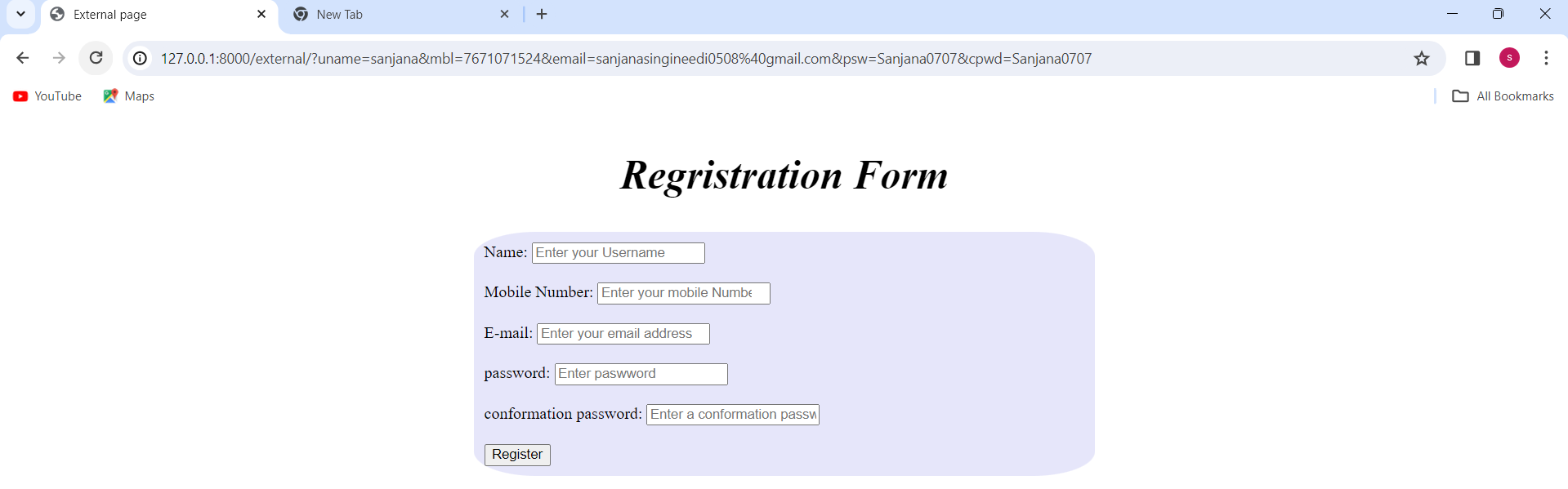
padding:10px;

border-radius: 10%;

}

**O/P:**

****

****

**Colour:** lavender

Some **styling for button** and **the code**:

button

{

background-color:yellow;

color:blue;

width:20%;

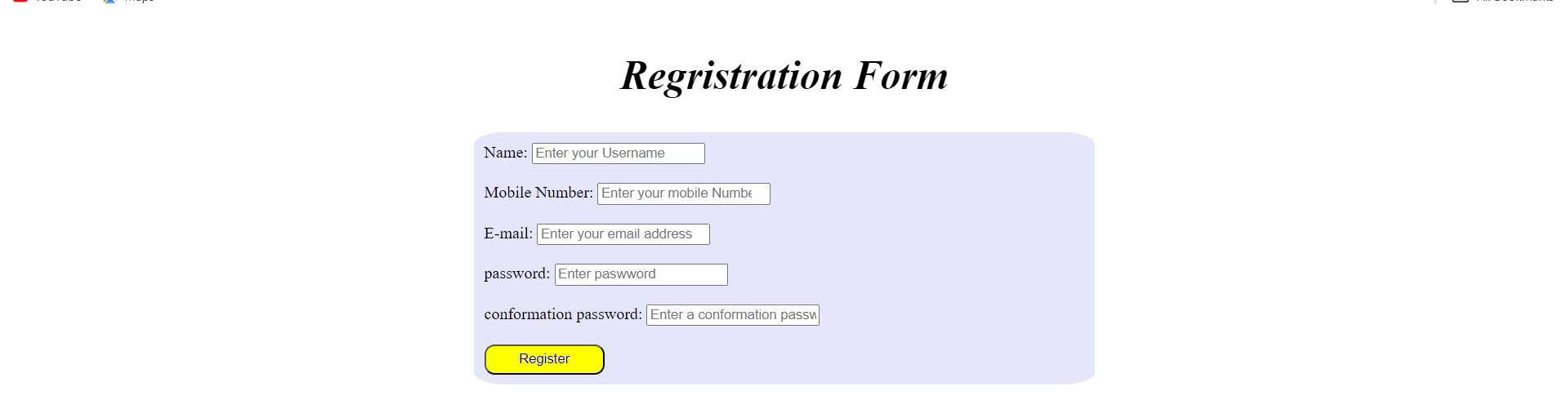
padding:5px;

border-radius: 10px;

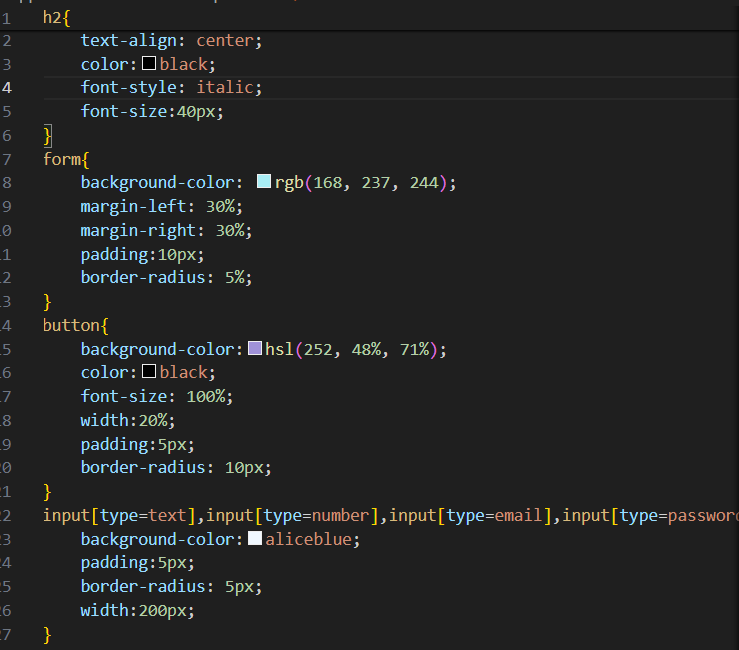
}

And change border-radius: 5px;

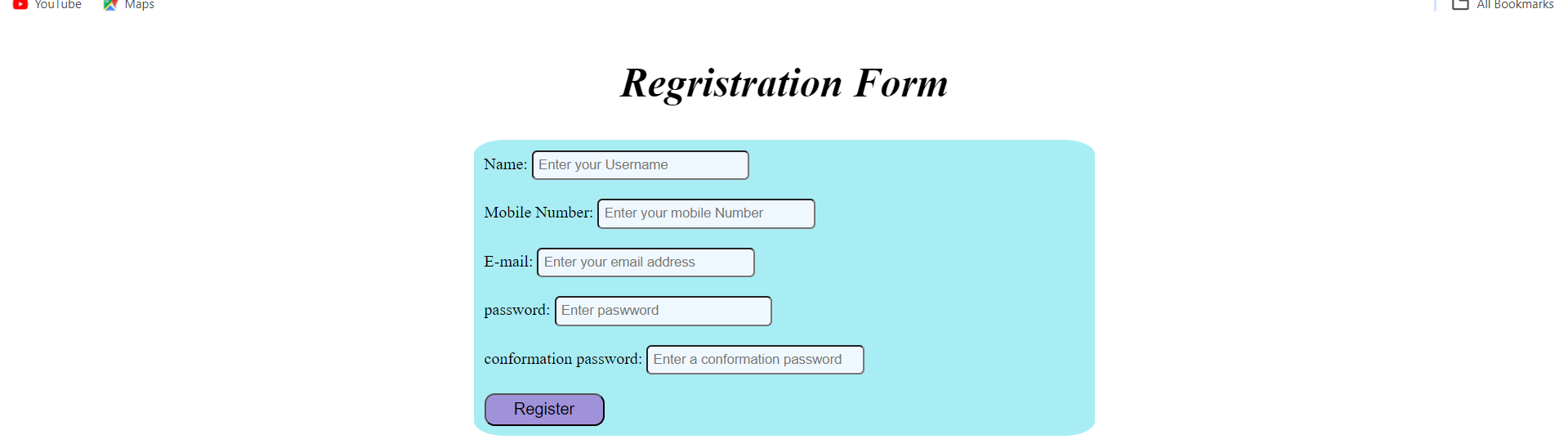
**O/P:**

****

**Code for css:**



**O/P:**



**Inserting the pic** in the background

**Save pic** in image file which is in **static folder with extension(**Like: .png, .jpg, .jpeg…..etc)

And code will be:

{% load static %}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1">

<title>External page</title>

<link rel="stylesheet" type="text/css" href="{% static 'css/sample.css'%}">

<style type="text/css">

body{

background-image: url("{% static 'images/pic1.jpg'%}");

background-repeat: no-repeat;

background-size: cover;

}

</style>

</head>

<body>

<h2>Regristration Form</h2>

<form>

Name: <input type="text" name="uname" placeholder="Enter your Username"><br><br>

Mobile Number: <input type="number" name="mbl" placeholder="Enter your mobile Number"><br><br>

E-mail: <input type="email" name="email" placeholder="Enter your email address"><br><br>

password: <input type="password" name="psw" placeholder="Enter paswword"><br><br>

conformation password: <input type="password" name="cpwd" placeholder="Enter a conformation password"><br><br>

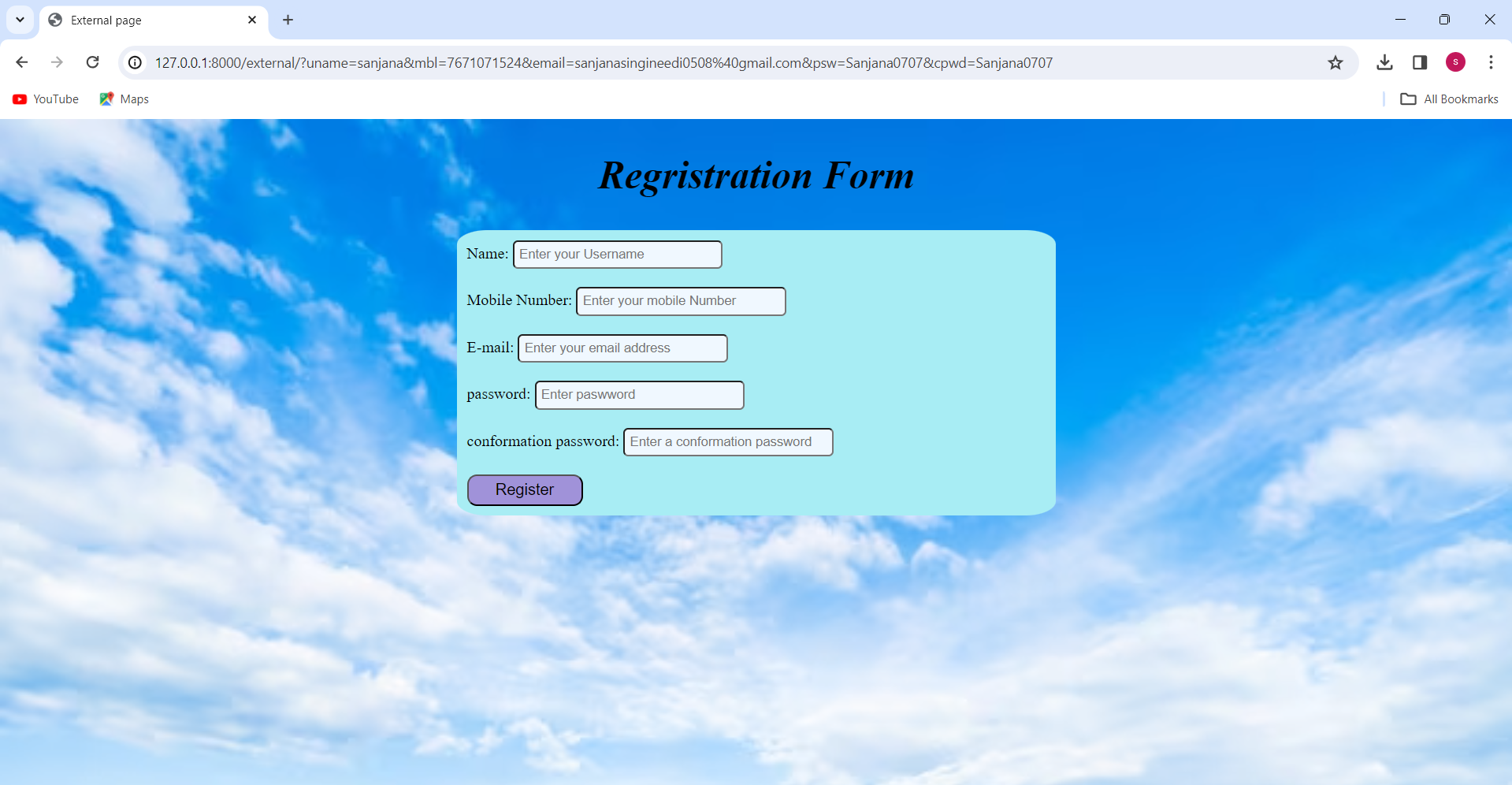
<button>Register</button>

</form>

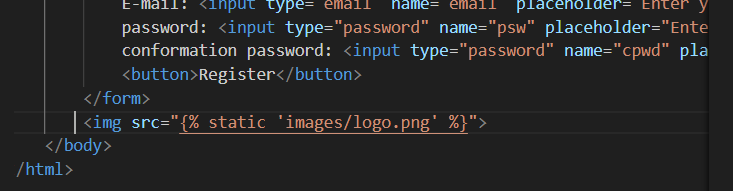
</body>

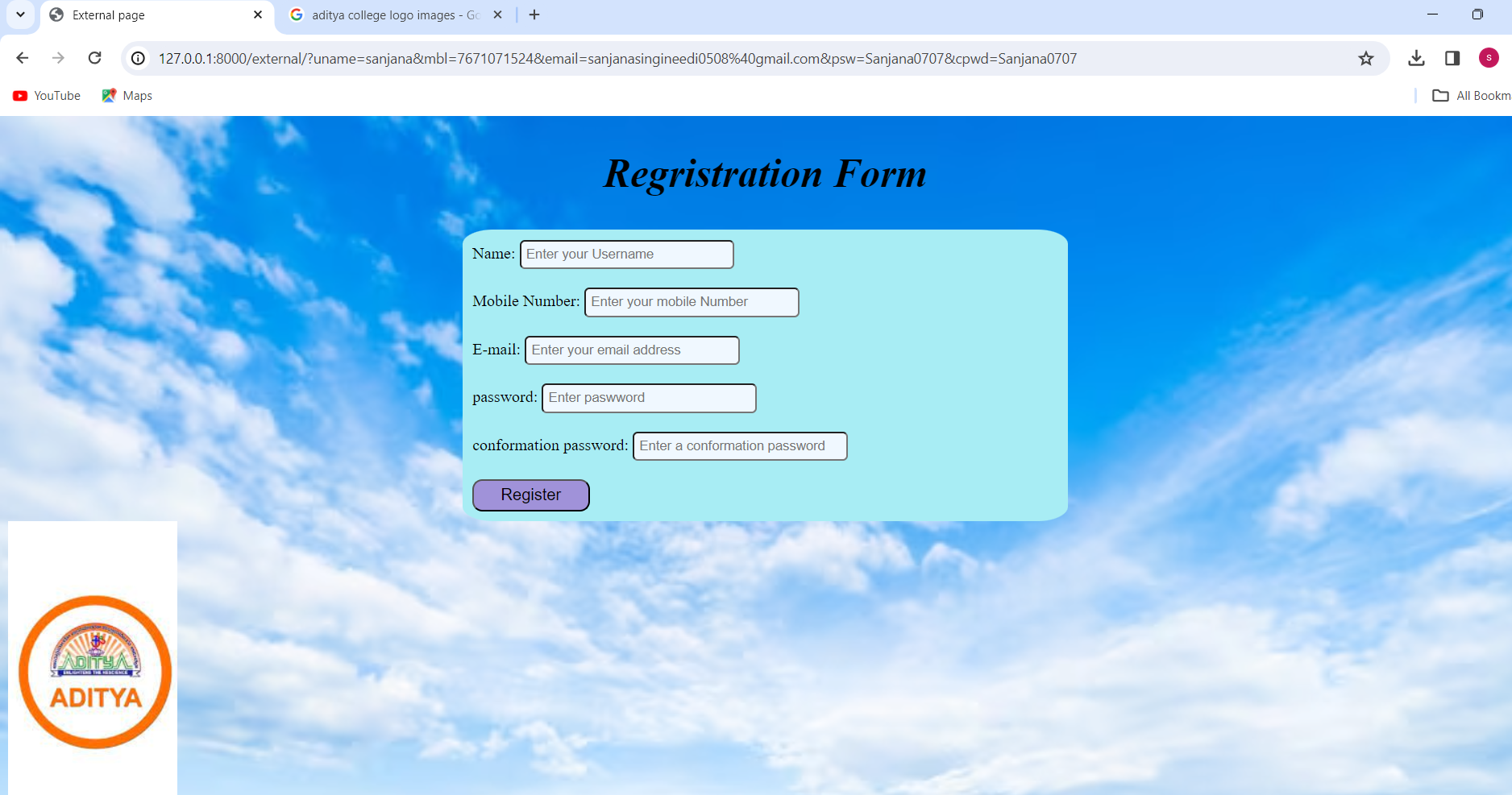
</html>

**O/P:**



**Adding image:**

**O/P:**

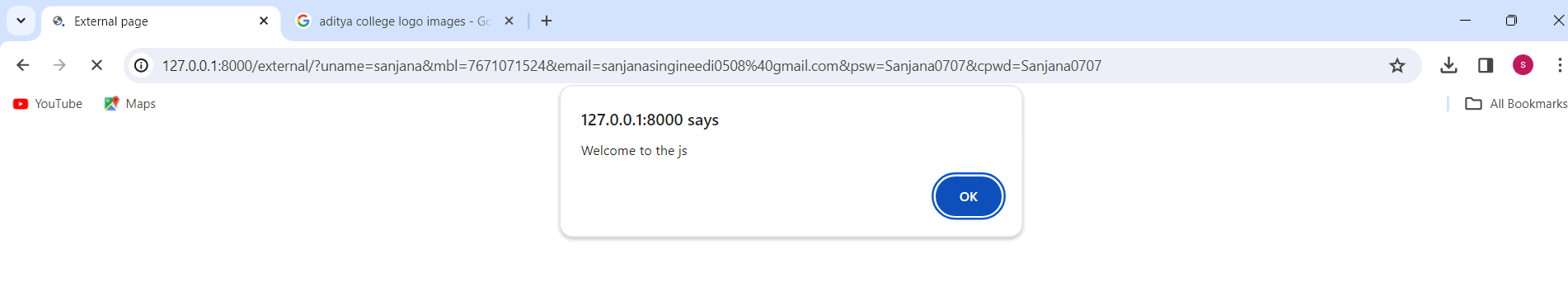


Now take one more file in the js folder which in static folder and save that new folder as **sample.js** (**javascript**) so in that js file write the below code:  
 alert("Welcome to the js")

In **external file** **before the style tag** and **after link tag**:

<script type="text/javascript" src="{% static 'js/sample.js' %}"></script>

**O/P:**



**DAY – 3**

**20-03-2024**

**Objectives :**  
 ->HTTP Request Methods

->Data Rendering from templates to views

->Django Admin

->Bootstrap

**HTTP’s Request methods:**  
We us e 2 types of request methods:

->GET method – default method

->POST method – In driver when we modify the page and navigate to another page we use

this method.

**Data Rendering from templates to views:**  
 user->server->urls->views->.html

**Extra navigations:** .html->views

To use post method we need some modifications like:

* .html-->Http method = ”POST” in form tag of external.html file
* If we want to provide the security for the data we use a token i.e **csrf\_token** (Security purpose).

**Views.py:** when we use post method we need to mention the functionality in the views file.

Condition need to mention is: if request. method ==”POST”:

Code in views file

def external(request):

    if request.method=="POST":

        name = request.POST['uname']

        mb=request.POST['mbl']

        em=request.POST['email']

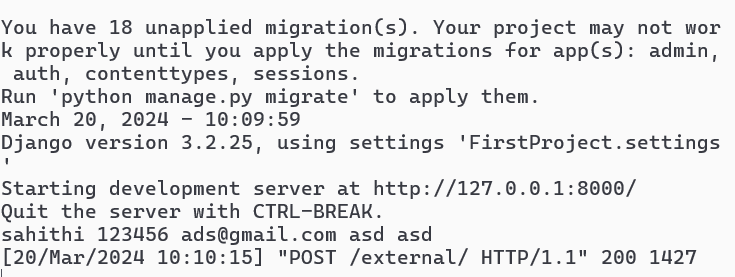
        ps=request.POST['psw']

        cps=request.POST['cpsw']

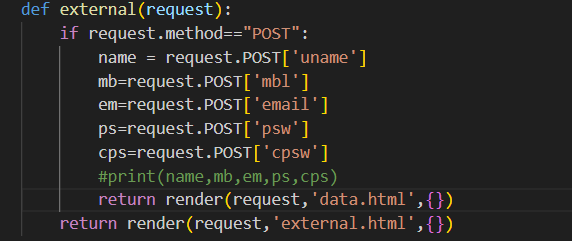
        print(name,mb,em,ps,cps)

    return render(request,'external.html',{})

Now fill the details in the form and submit it then the details will be displayed in the server.



The output displays on the server to display it separately we write a render function instead of the print statement as shown in below.



Create a data.html new file in templates and write the below code in that html file:  
<html>

    <head>

        <meta charset="utf-8">

        <meta name="viewport" content="widht=device-width,initial-scale=1">

        <title>Regristration data </title>

    </head>

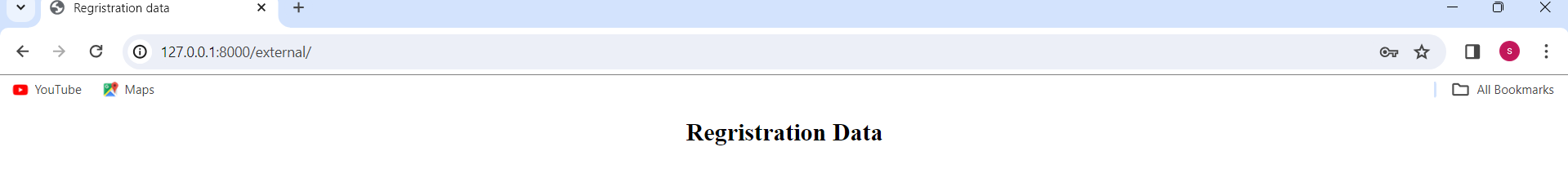
    <body>

            <h2 style="text-align: center;">Regristration Data</h2>

    </body>

</html>

Save and fill data in the data in the output and submit then it navigates to the a another page as shown below:



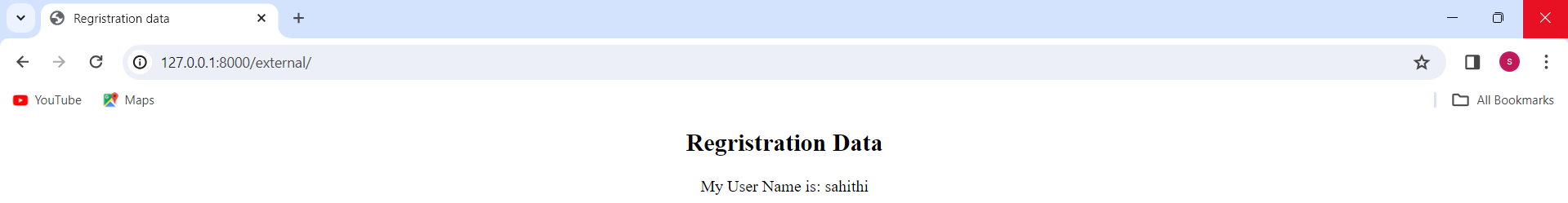
**To pass the data from view to html file** we use the **dictionary** by assign the values to the names of attributes like :

return render(request,'data.html',{'n':name})

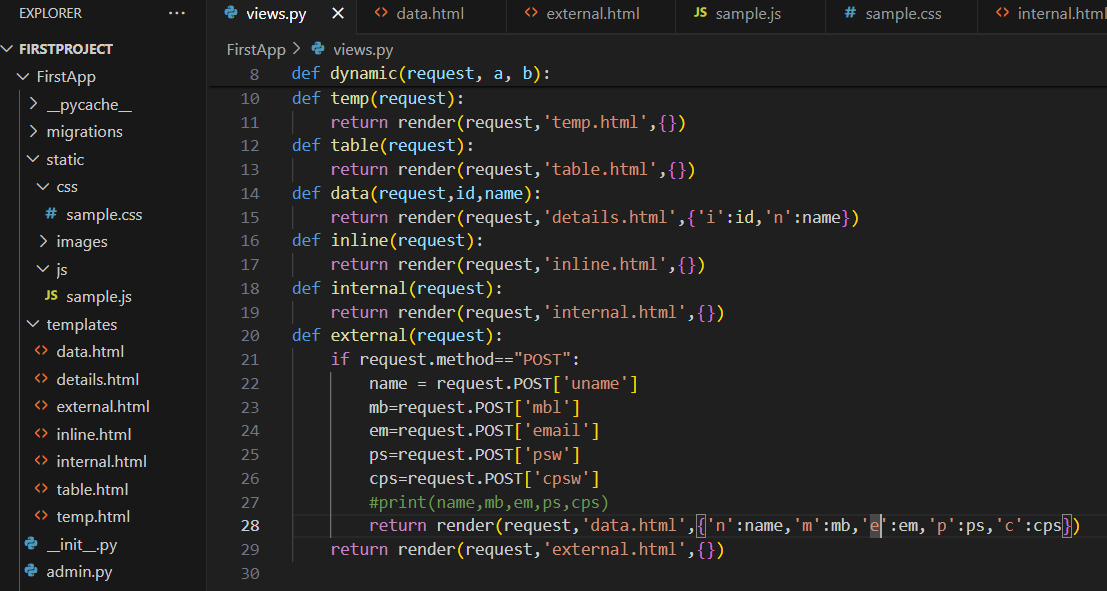
In **data.html file** update the code by:

<p style="text-align: center;"> My User Name is: {{n}}</p>

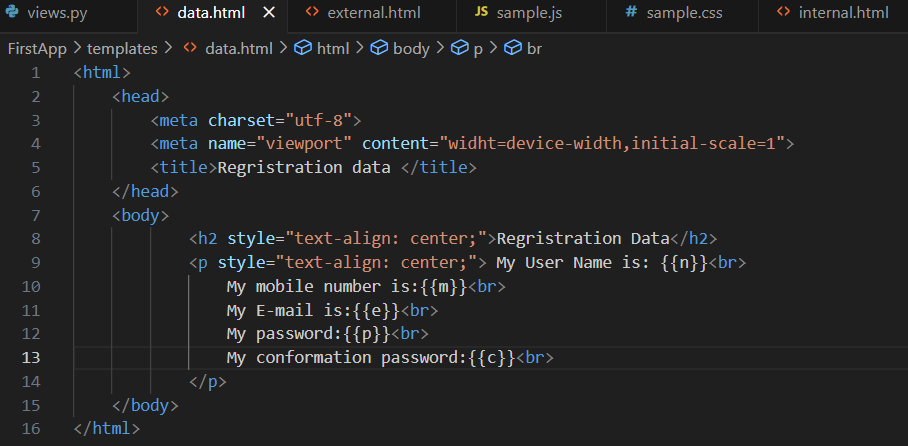
**The O/P:**

****

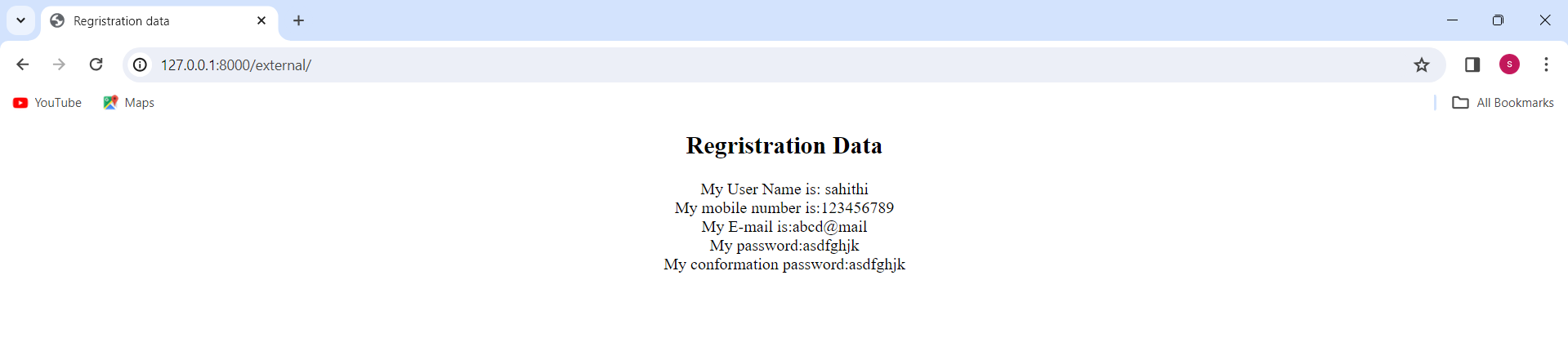
**The code to display the whole data:**



**In data.html file:**



**O/P:**



Now we going to print the above **O/P in the table format**, so we made some changes in the data.html file as shown below:

Code in data.html file is:  
<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>Regristration data </title>

</head>

<body>

<h2 style="text-align: center;">Regristration Data</h2>

<center>

<table border="4" width="50%">

<thread>

<tr>

<th>UserName</th>

<th>Mobile Number</th>

<th>E-mail</th>

<th>password</th>

<th>conformation Password</th>

</tr>

</thread>

<tbody>

<tr>

<td>{{n}}</td>

<td>{{m}}</td>

<td>{{e}}</td>

<td>{{p}}</td>

<td>{{c}}</td>

</tr>

</tbody>

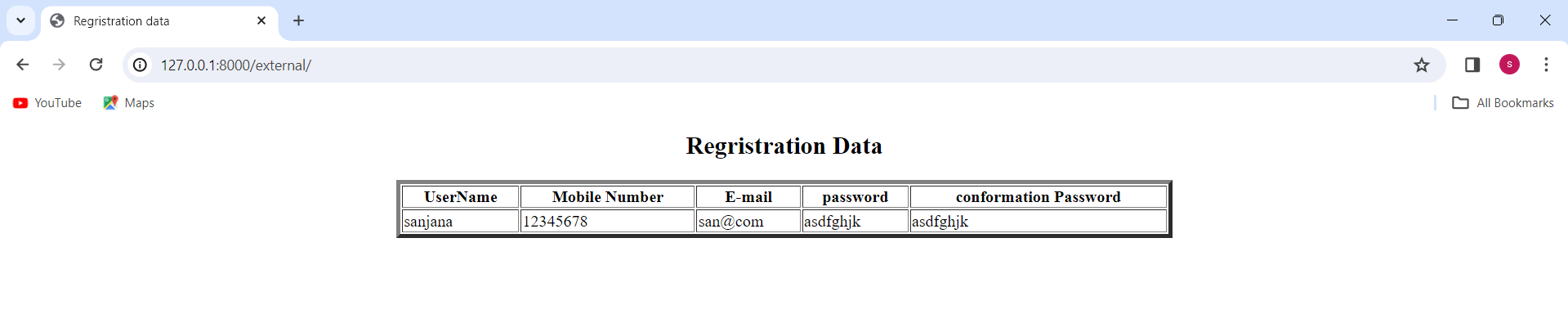
</table>

</center>

</body>

</html>

**O/P:**



**Django Admin:**

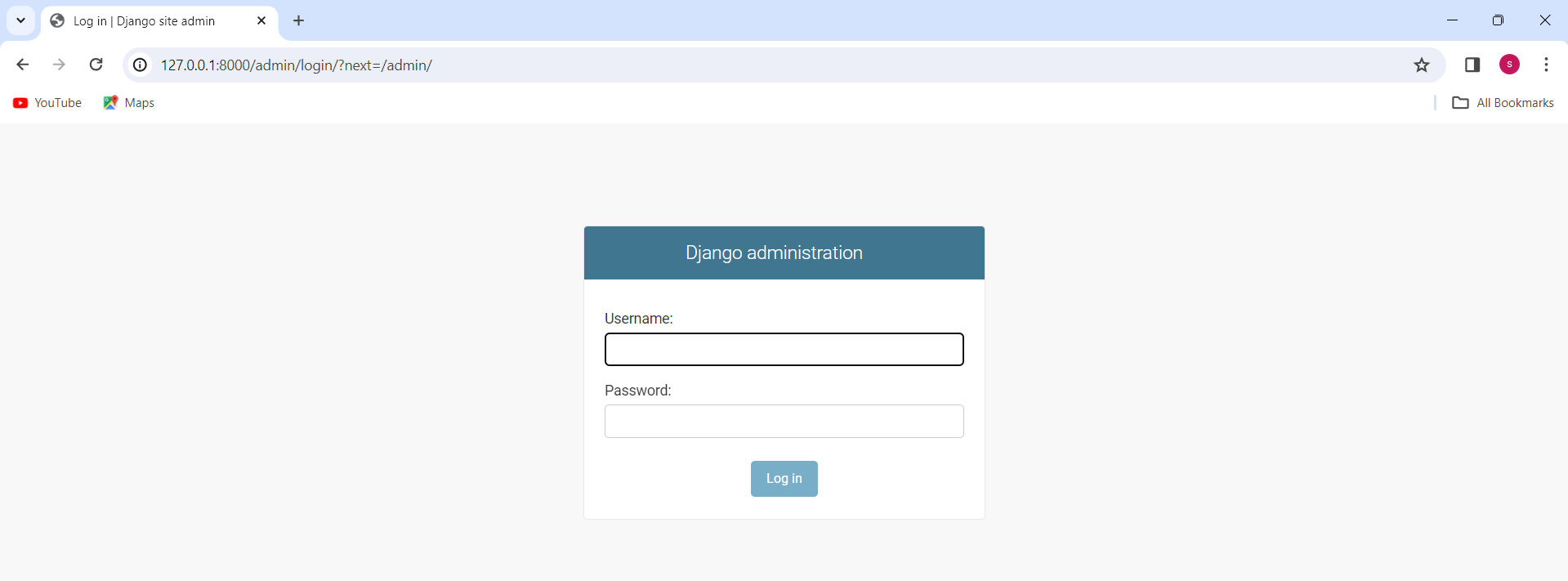
We are having the default url that is admin url:

Django administration GUI

**https://127.0.0.1:8000/admin**

path('admin/', admin.site.urls),

O/P:

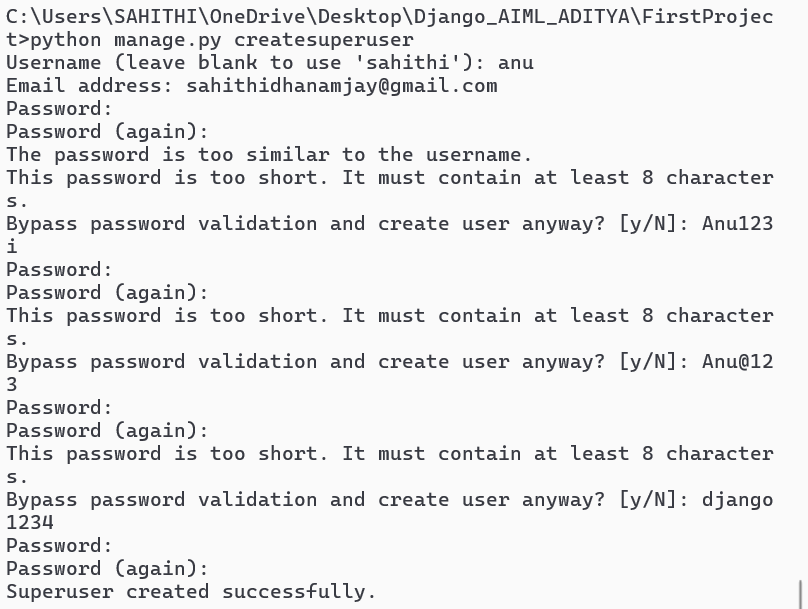


Python manage.py migrate – There are default table in Django server to display that in the admin page we use this command and then we can create our super user account.

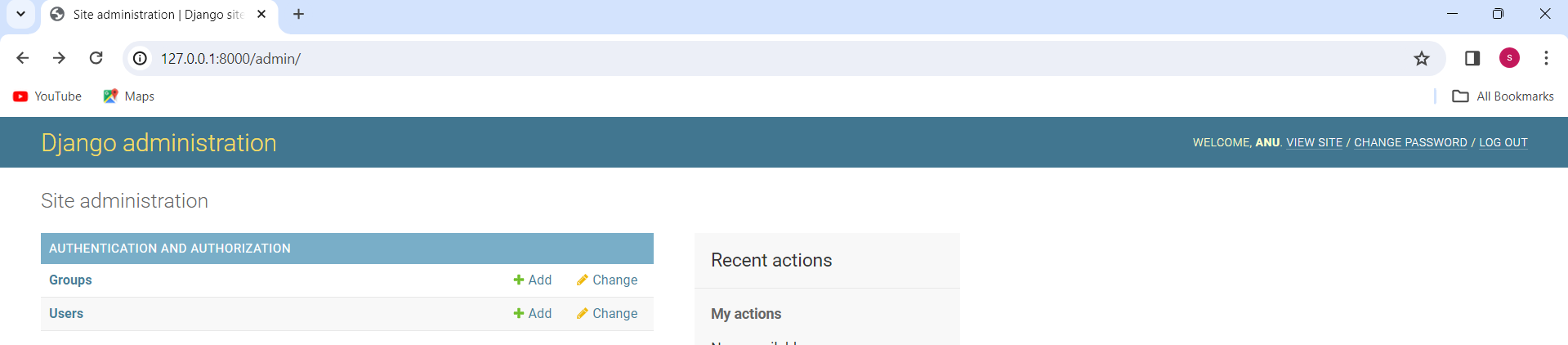
**superuser account:**

python manage.py createsuperuser

then the command prompt will give:

****

**After this in the above admin page use the created username and password in the administration page and we get**



Here we can add the user by user option->add user->enter username, password, conform password and then access page.

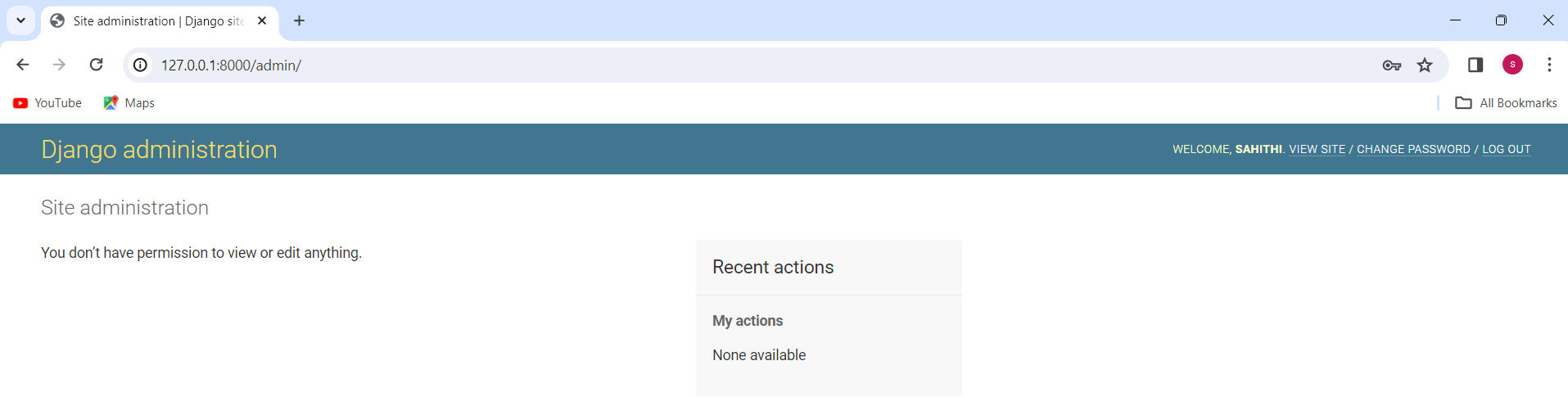
In adding a person the admin also have a assigning the control powers for the going to add member like staff, super. In super access the person can access like a admin that means both are having the equal powers (admin and super user) where as staff access we are not having t=such access.

anu - django1234

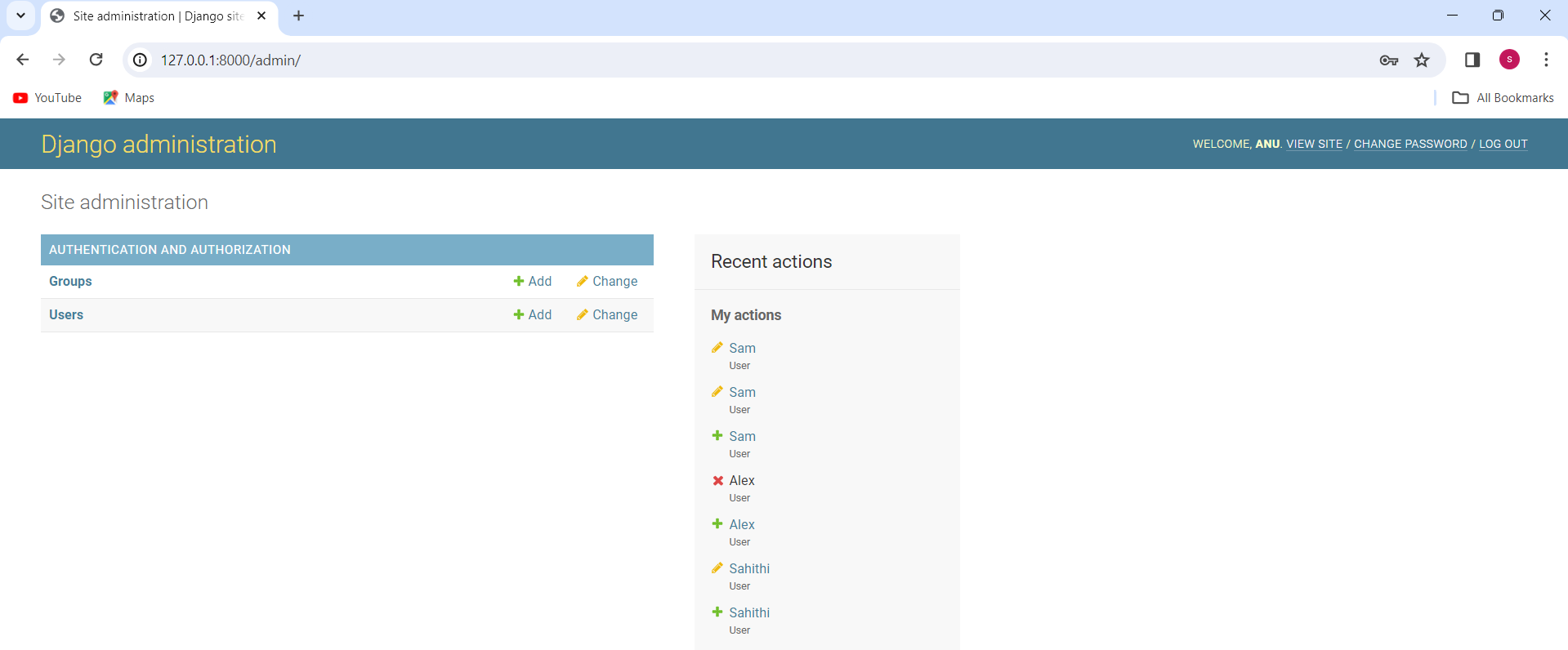
sahithi- sanjana1234

sam-SAMsam123

**Staff access:**



For **super access** and **admin access** will be



**Bootstrap:(** <https://getbootstrap.com/>**)**

**Bootstrap** is the most popular **CSS Framework** for developing responsive and mobile-first websites. It is used to provide the **efficient styling** to the webpage. Create **a new url** with name boot and a **function** in views.py and a **template as boot.html**.

**New URL creation:**

path('boot/',views.boot,name="boot")

**new function in:**

def boot(request):

    return render(request,'boot.html')

Boot strap is applied in **2 different ways:**

**Online bootstrap -** we can copy both css and js links from the online

**Syntax:  
 <property\_name>-value**

Used website to copy the links from online is **(** <https://getbootstrap.com/>**)** In that referred link the **Link tag** indicates – css file and **script tag** indicates **js** files. We take the links from the download option from the above link and then paste after the title in theboot.html and refresh the page where we get the difference in the font style in the output.

**Boot.html code:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>BootStrap page</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>

</head>

<body>

<div>

<div>

<div>

<h2>Login Form</h2>

</div>

</div>

</div>

</body>

</html>

By using class attribute we can apply bootstrap properties to html elements or html tag. To see the color code in the bootstrap which can be seen in the customize section in that color and by default 8 colors.

<h2 class="bg-Warning">Login Form</h2>

**O/P:**



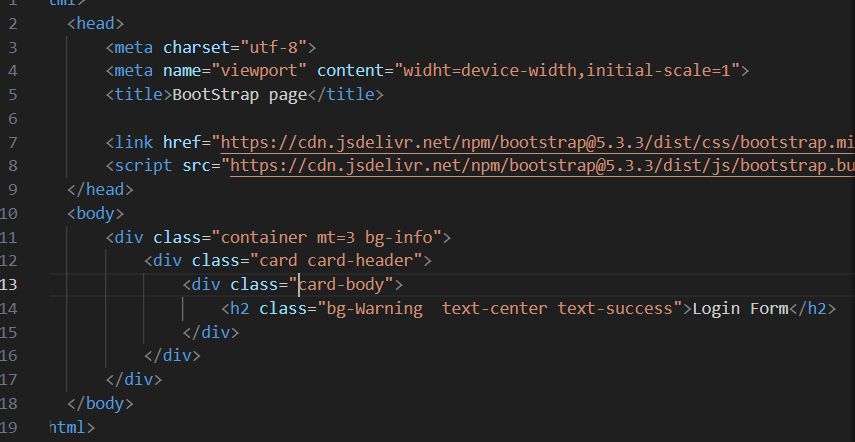
Now we need to move the LOGIN FORM heading to center and change its color:

<h2 class="bg-Warning  text-center text-success">Login Form</h2>

**O/P:**



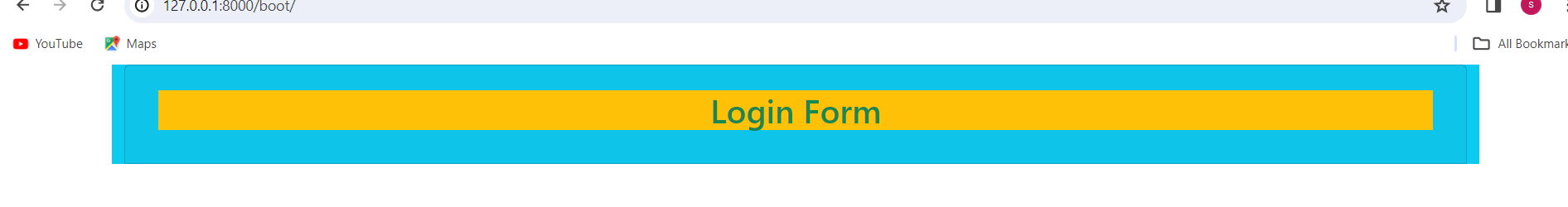
To differentiate the above search bar and your title we use the following code:



Mt-margin top max possible value is :5

Container- container tags are HTML elements we use to create a section or a box on a web page

Card-A card is a flexible and extensible content container

**O/P:**  


To create a login page we create a 2 inputs texts and if we need to increase the textbox size then use class=”form-control”.

**Code:**

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="widht=device-width,initial-scale=1">

<title>BootStrap page</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"

rel="stylesheet" integrity="sha384-

QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALE

wIH" crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-

YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeH

z" crossorigin="anonymous"></script>

</head>

<body>

<div class="container mt=3 bg-info">

<div class="card card-header">

<div class="card-body">

<h2 class="bg-Warning text-center text-success">Login Form</h2>

<form>

<input text="text" name="uname" placeholder="Enter your username"

class="form-control">

<input text="password" name="psw" placeholder="ENter your passsword">

</form>

</div>

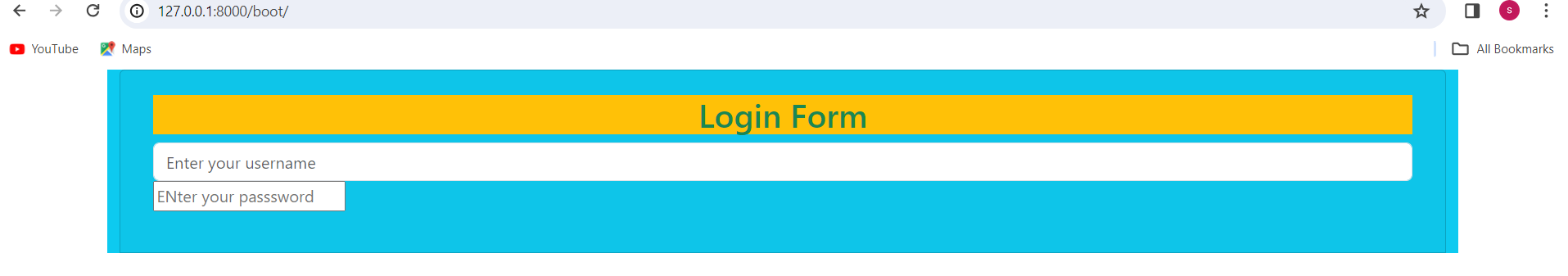
</div>

</div>

</body>

</html>

**O/P:**

****

**To give the space in between text file we use:**

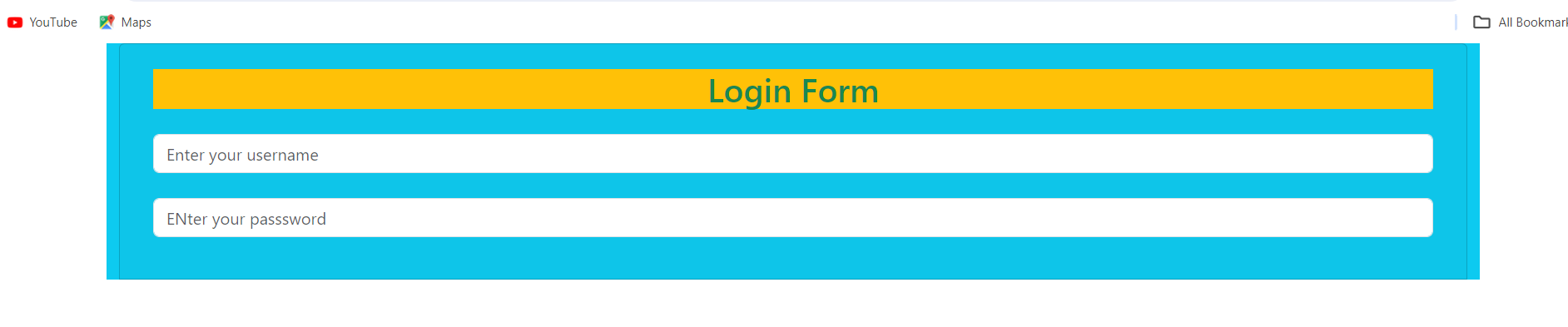
<form>

    <input text="text" name="uname" placeholder="Enter your username"

                        class="form-control my-4">

<input text="password" name="psw" placeholder="ENter your passsword"

                        class="form-control">

**O/P:  
**

**Button:**

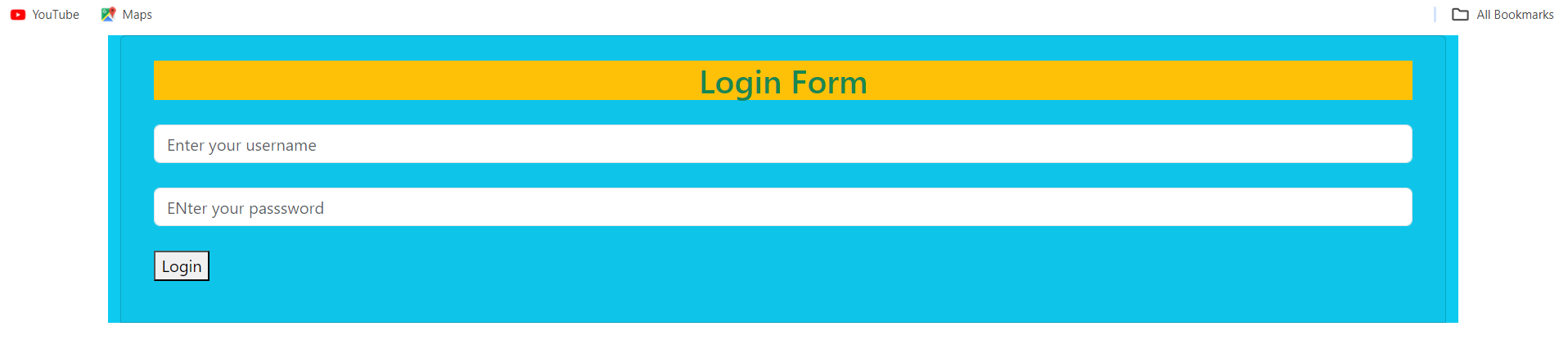
 <input text="text" name="uname" placeholder="Enter your username"

  class="form-control my-4">

 <input text="password" name="psw" placeholder="ENter your passsword"

                        class="form-control my-4">

                        <button>Login</button>

****

**If we want to mention button in centre then**

 <form>

        <input text="text" name="uname" placeholder="Enter your username"

         class="form-control my-4">

         <input text="password" name="psw" placeholder="ENter your passsword"

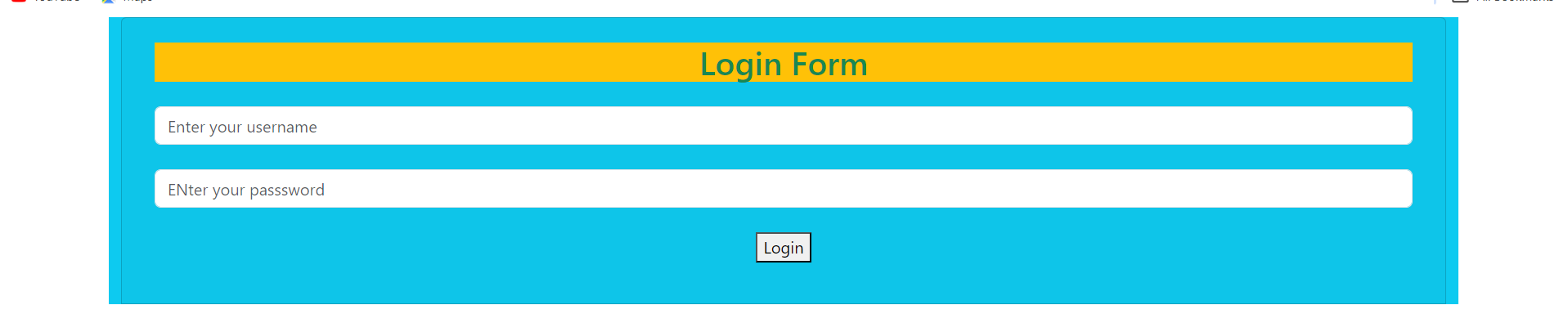
                        class="form-control my-4">

                        <div class="text-center">

                            <button>Login</button>

                        </div>

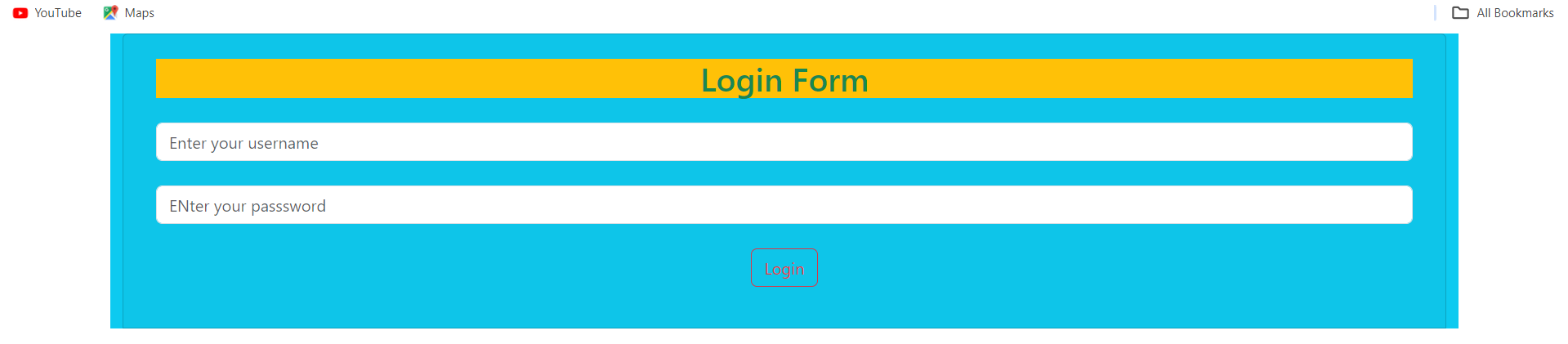
                    </form>

****

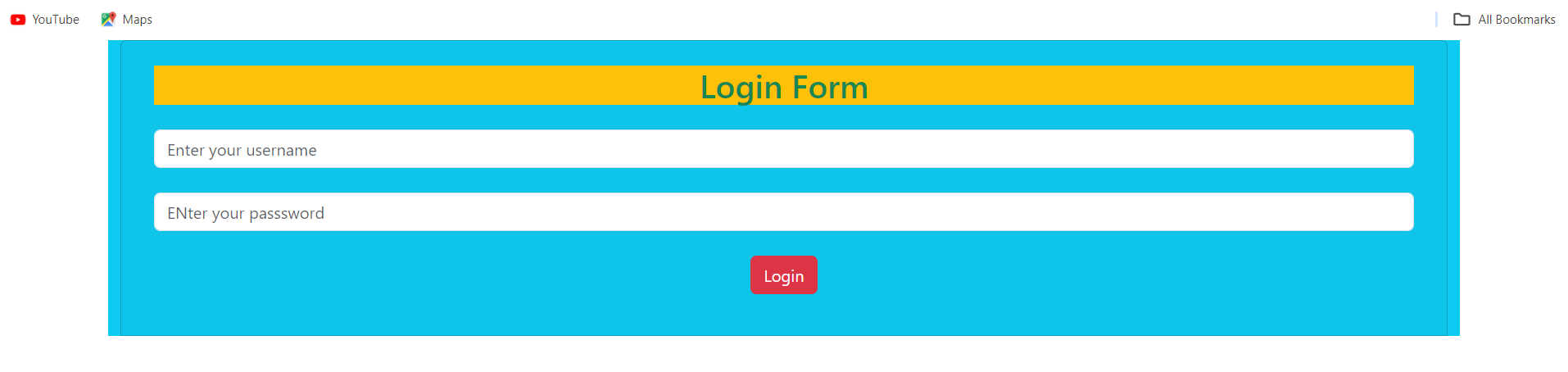
**Some modification for button**:

In bootstrap **btn** is responsible for the colour of the button.

<button class="btn btn-outline-danger">Login</button>

****

We the cursor is on the button we get:

****

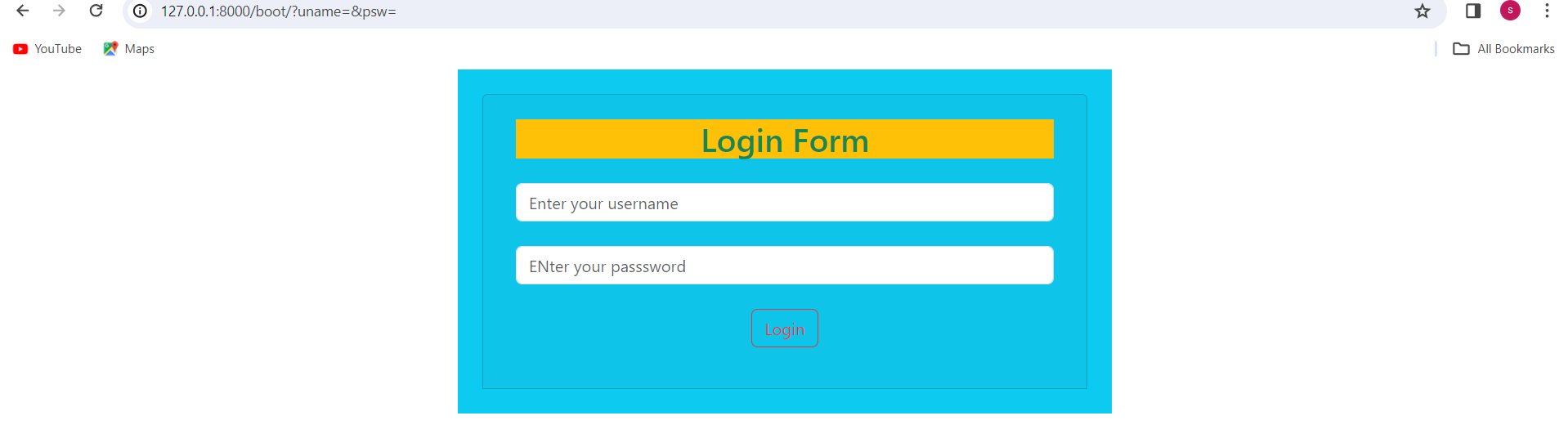
We can also use padding to remove the left side and right-side space we get:

In div tag do this modification:

 <div class="container mt=3 bg-info p-4 col-md-5">

**p =** padding, **md** = margin-down, **bg** = back-ground colour, **mt** = margin-top

**O/P:**

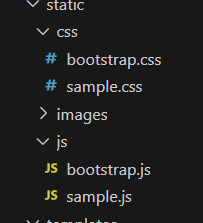
****

**Offline bootstrap:**

Here we can store both css and js links in our local system or pc. Mostly we use offline bootstrap, we save the link in the local system that is in PC.

To see that, we copy the href link present in the link tag and check it by pasting in the searching bar in the browser and we get bootstrap folder. Now we save that in static->css-> save with name: bootstap.css and we get the update in the visual studio code.

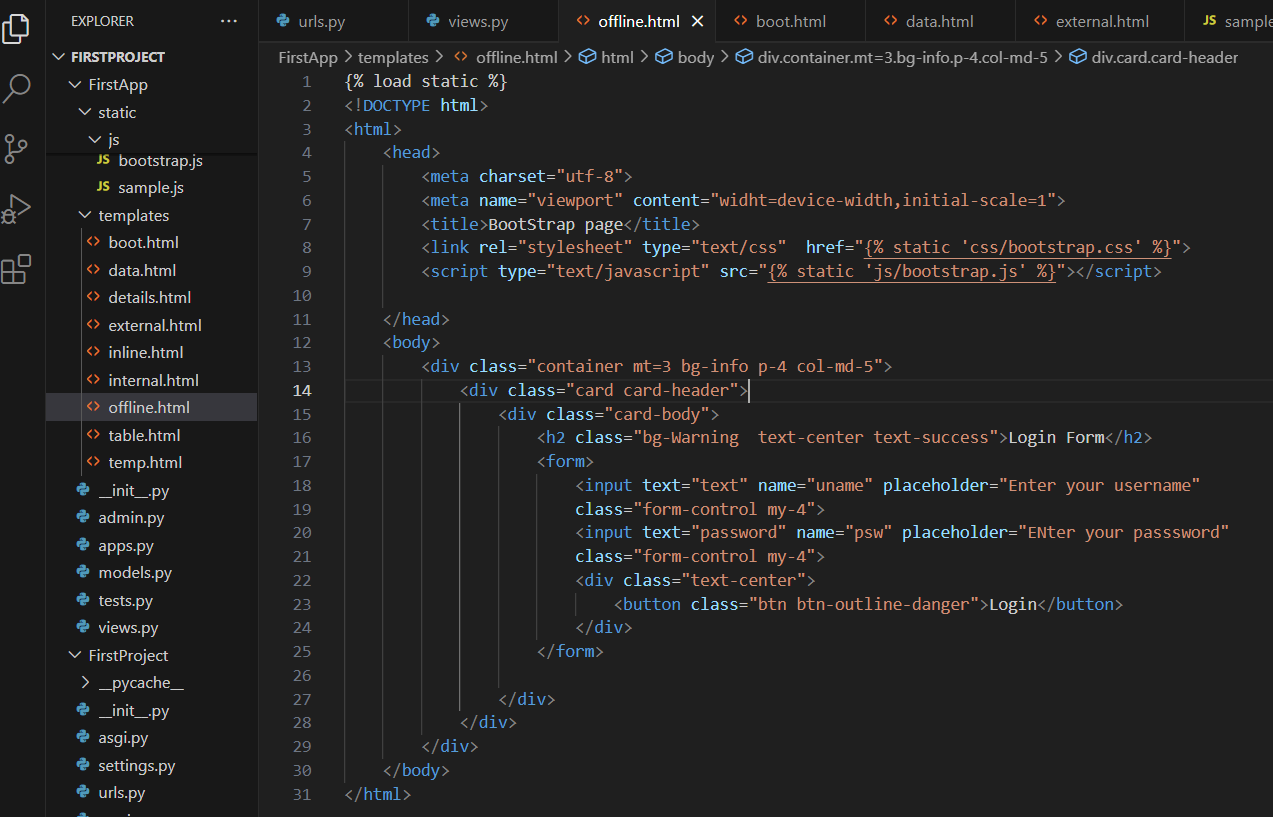
Perform the same action for the js for too that is copy href of script tag, paste in browser, save in static-js folder save with the name: bootstrap.js and update take place in the visual studio code

****

**Create a new URL and function and offline.html:**

 path('offline/',views.offline,name="offline")

Now create a bootstrap.css in css folder which is in static and bootstrap.js in js folder which is present in js file. Then we use the below code:

**Offline.html** code will be:  


**O/P:**  


**DAY-4**

**21-03-2024**

**Objectives:**

**Task:**

**Create with a name of register:**

Registration form by **offline bootstrap** and input fields:

- Username

- email

- mobile number

- radio buttons (Male or Female)

- check box (Languages – Telugu, English, Hindi)

- Drop-down (branches name -CSE, ECE, AI, CSI, MECH)

- joining date

- update your resume

- password

- conform password (By using offline bootstrap).

**Working:**

**Reference:(** <https://getbootstrap.com/>**)**

**Creating a new URL:**

path('register/',views.register,name="register")

**Function:**

def register(request):

    return render(request,'register.html')

**register.html:**

{% load static %}

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

        <meta name="viewport" content="widht=device-width,initial-scale=1">

        <title>BootStrap Register page</title>

        <link rel="stylesheet" type="text/css"  href="{% static

'css/bootstrap.css' %}">

        <script type="text/javascript" src="{% static 'js/bootstrap.js'

%}"></script>

    </head>

    <body>

        <div class="container mt=3 bg-info p-4 col-md-5">

            <div class="card card-header">

                <div class="card-body">

                    <h2 class="bg-Warning  text-center text-success">Login

Form</h2>

                    <form>

                        <input text="text" name="uname" placeholder="Enter

your username" class="form-control my-4">

                        <input text="number" name="mbl" placeholder="Enter

your mobile number:" class="form-control my-4">

                        <input text="text" name="email" placeholder="Enter

your E-mail" class="form-control my-4">

                        <input class="form-check-input" type="checkbox"

value="" id="flexCheckDefault">

<label class="form-check-label" for="flexCheckDefault">Telugu</label>

                     <input class="form-check-input" type="checkbox" value=""

id="flexCheckDefault">

                     <label class="form-check-label"

for="flexCheckDefault">Hindi</label>

                     <input class="form-check-input" type="checkbox" value=""

id="flexCheckDefault">

                     <label class="form-check-label"

for="flexCheckDefault">English</label><br><br>

                     <input class="form-check-input" type="radio"

name="flexRadioDefault" id="flexRadioDefault1">

                     <label class="form-check-label"

for="flexRadioDefault1">Male</label>

                     <input class="form-check-input" type="radio"

name="flexRadioDefault" id="flexRadioDefault1">

                     <label class="form-check-label"

for="flexRadioDefault1">Female</label>

                     <br><br>

                     <button class="btn btn-secondary dropdown-toggle"

type="button" data-bs-toggle="dropdown" aria-expanded="false"

>Branches</button>

                     <ul class="dropdown-menu">

                            <li><a class="dropdown-item" href="#">CSE</a></li>

                            <li><a class="dropdown-item"

href="#">AIML</a></li>

                            <li><a class="dropdown-item" href="#">ECE</a></li>

                            <li><a class="dropdown-item" href="#">CSI</a></li>

                            <li><a class="dropdown-item"

href="#">MECH</a></li>

                          </ul>

                          <br><br>

                      <input text="text" name="resume" placeholder="Update

resume" class="form-control my-4">

                      <input text="password" name="psw" placeholder="ENter

your passsword" class="form-control my-4">

                      <input text="password" name="psw" placeholder="ENter

your conformation passsword"class="form-control my-4">

             <div class="text-center">

          <button class="btn btn-outline-danger">Login</button>

              </div>

               </form>

                </div>

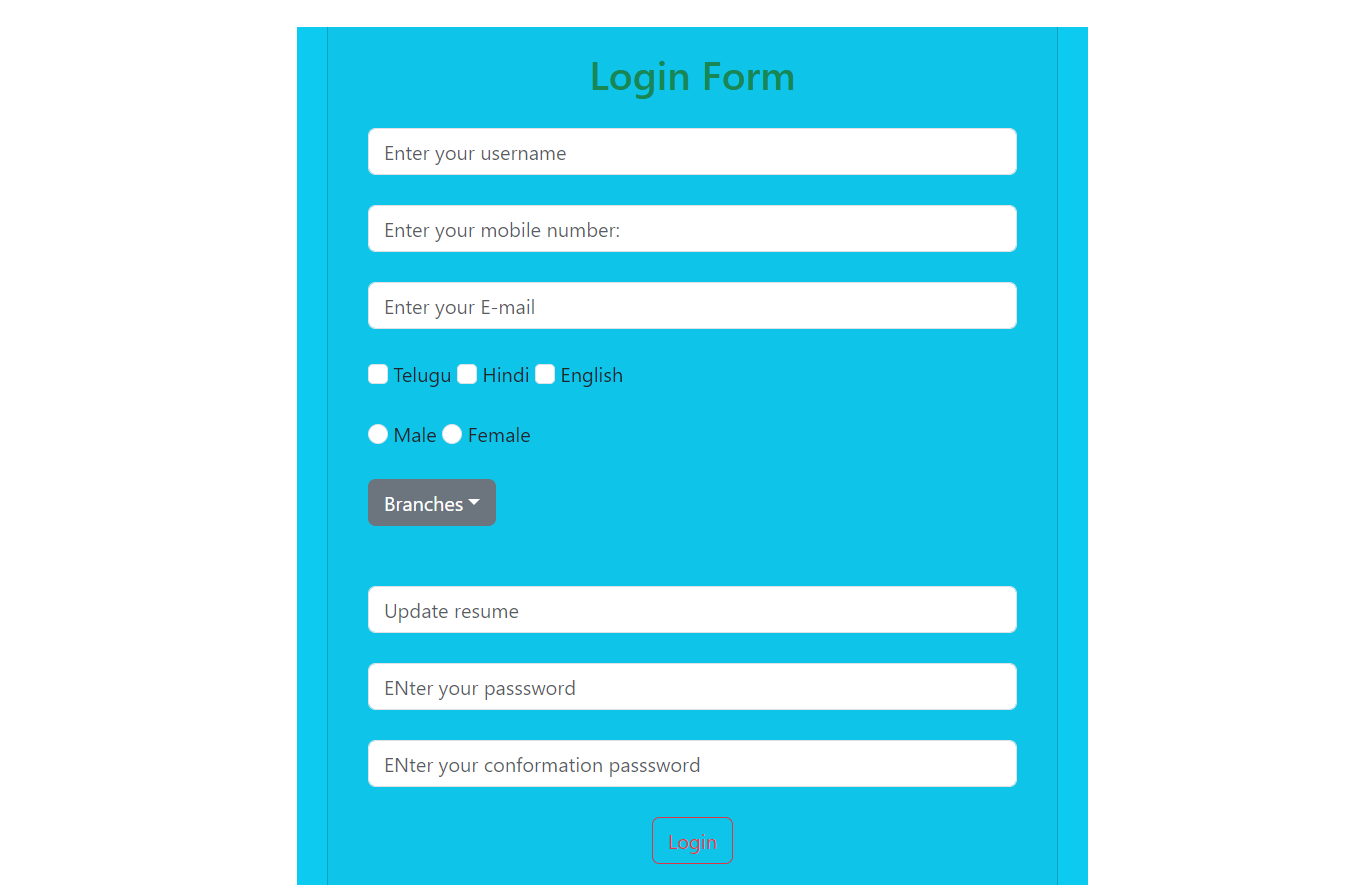
            </div>

        </div>

    </body>

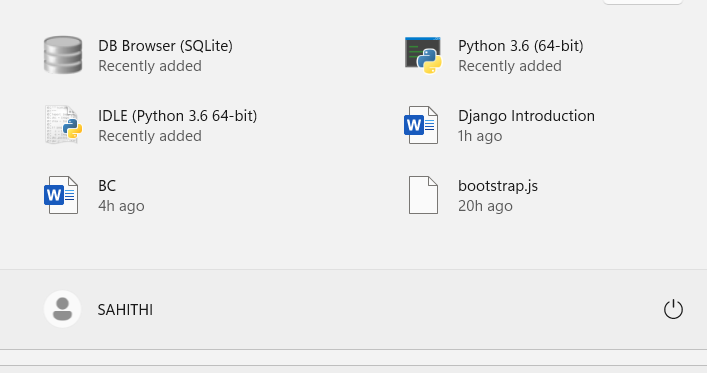
</html>

**O/P:**

`

**Model Creation**

We need to download and install the data base that is **sqlite**(https://sqlitebrowser.org/dl/) and download: [DB Browser for SQLite - Standard installer for 64-bit Windows](https://download.sqlitebrowser.org/DB.Browser.for.SQLite-3.12.2-win64.msi) Click on this for download. Now open the downloaded file and choose the sqlite option and install we get the option in the menu as shown in below:



**Model:**

* Model is used to store the database files.
* To create a data base we use python classes in Django we use a file that is models.py file.
* Open models.py file visual Studio code:

from django.db import models

Here Django-package and db-module

* we are going to create a database: database name: Student:
  + - name
    - roll num
    - age
    - mobile
    - email
    - address

**Syntax:**  
 class classname -->class name act as a database name

In models function we use attribute that is model

In Django we use data filed that is:

String – **CharField()**

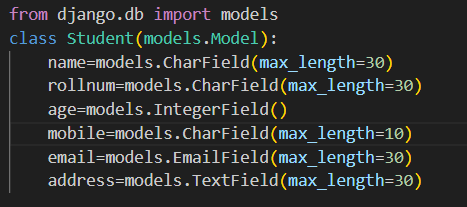
Integer – **IntegerFiled()**

Email – **EmailField()**

Large text – **TextField()**

Image – **imageField()**

To insert the data into database we need to enter the maximum length of the input:



We create a database and if need that to be converted into a table then after completion of model creation we have to perform 2 operations:

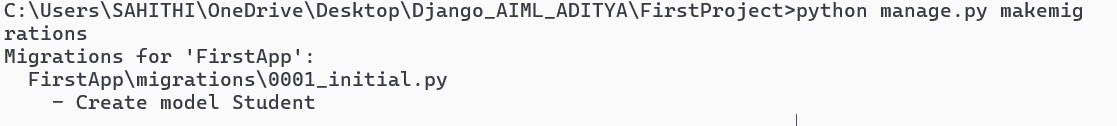
* make migrations
* migrate

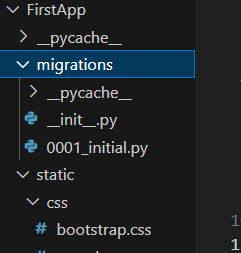
**Make Migrations:** It generates one interface file for database we use a command that is:

Python manage.py makemigrations

Then a interface file is downloaded and created a separate folder in the vs code.

**O/P:**

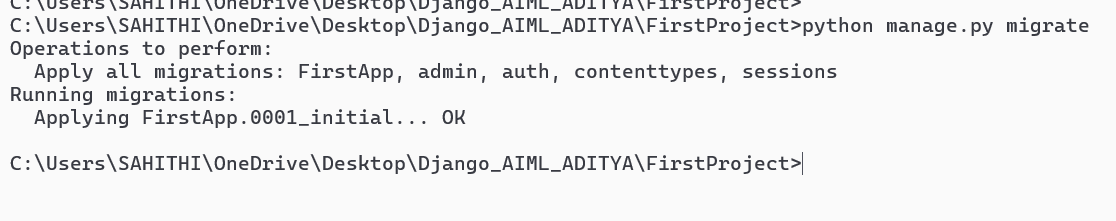
****

****

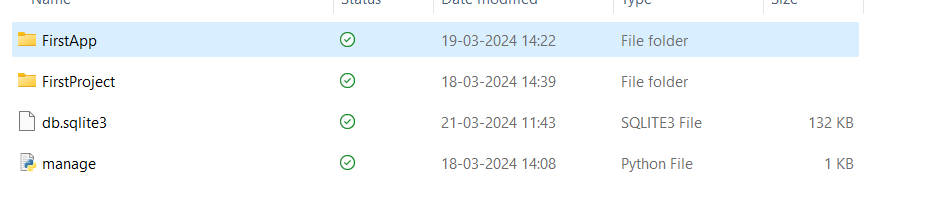
**Migrate-** It converts the interface file into the table.

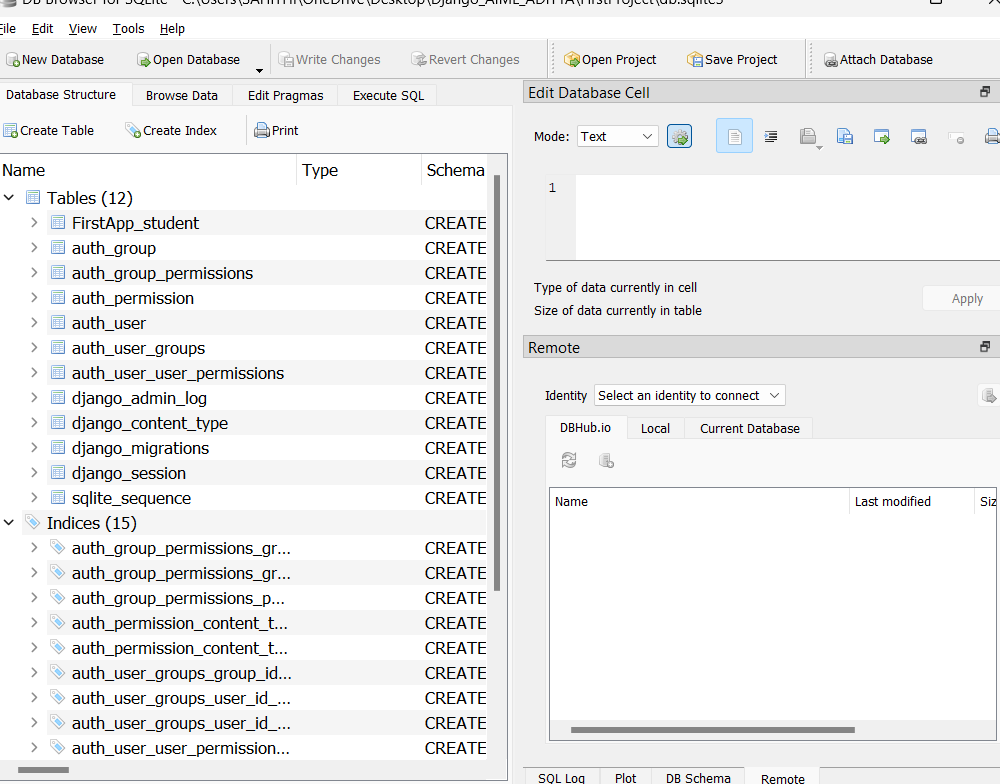
**Syntax:**  
 python manage.py migrate

**O/P:**

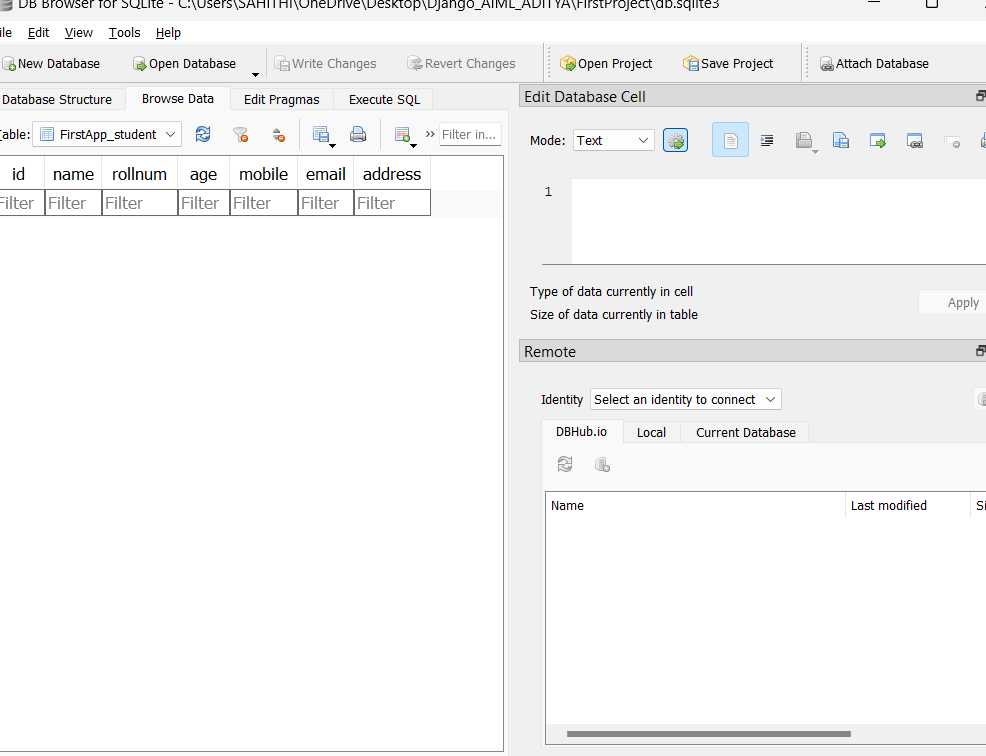
****

Open db browser sql and we are having a file in folder, in file->OpenDataBase->choose folder->choose db.sqlite3 file and those table present in it will be displayed as the following

****

****

Now **to display that in the table** format we go with browse option and we get:



If we want to insert or retrieve the data from the database then we use the separate queries that is ORM Queries:

**ORM Queries**

* Insert, Read, Update, Delete operation can be performed by these queries.
* We use python shell and to open the shell the command is:

**Syntax:**

python manage.py shell

* Then python interactive shell open and the commands will work only in the python shell.
* Need to import the database into the shell by giving the following command:

**Syntax:**

from FirstApp.models import Student

**How to insert data into database by using ORM Queries?**

To insert any data into the database we are having 2 different methods:

* **Save()**

**Query:**

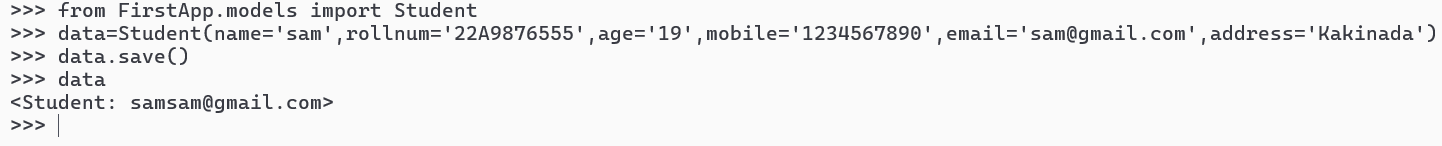
data=Modelname(name=’xxxx’,rollnum=’xxx’,age=’xxx’,mobile=’xxxxxxx

’,email=’xxxx@gmail.com’,address=’xxxx’)

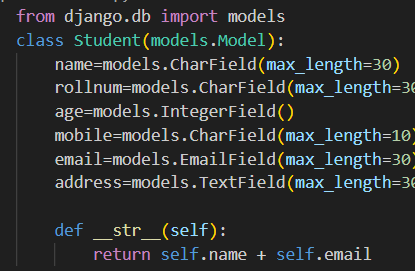
data.save() ->Need to save the query

data

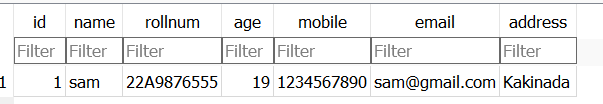
**Execution:**



Here name + email will be displayed as mention those in the return



When we refresh then those values will be added to database,

****

Adding some more data into database:

**Commands given in command Prompt:**

data=Student(name='alex',rollnum='22A901344',age='20',mobile='9876543021',email='alex@

gmail.com', address='America')

data.save()

data

**O/P:** <Student: alexalex@gmail.com>

data=Student(name='sri',rollnum='22A94344',age='23',mobile='9222543021',email='alesri@g

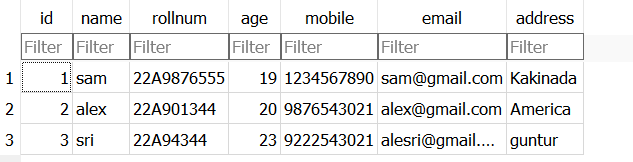
mail.com',address='Guntur')

data.save()

data

**O/P:** <Student: [srialesri@gmail.com](mailto:srialesri@gmail.com)>

**O/P:**

O/P:  


* **Create()**

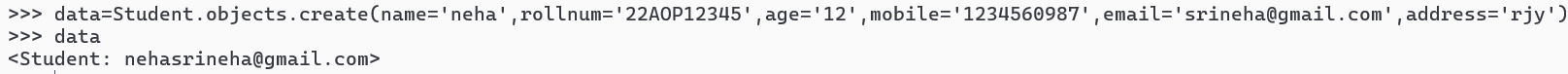
**Query:**

data= Modelname.Objects.create(name=’xxxx’,rollnum=’xxx’,age=’xxx’,

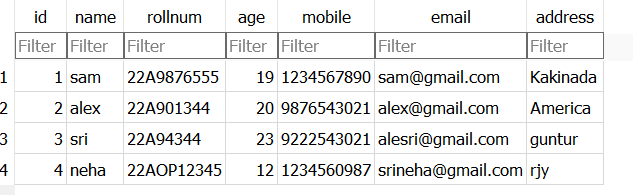
mobile=’xxxxxxx’,email=’xxxx@gmail.com’,address=’xxxx’)

data

**Execution:**

****

Thus it had also be updated in the db-sqlite table:



**How to retrieve the data from the database by using ORE queries:**

If we want to retrieve the total data from the database here we are using:

* **all()**

**query:**

modelname.objects.all()

The **Output** is in the form of list

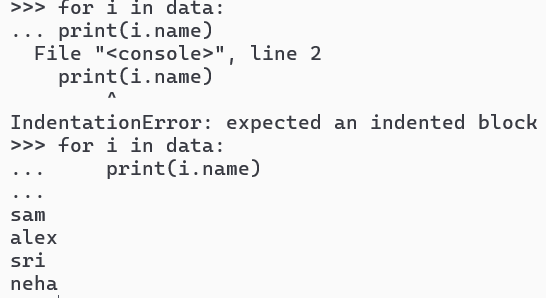


For suppose if we want to **access the single column** data from the database we can **apply the for loop** for that variable

for i in data:

print(i.columnName)

**Execution:**

****  ****

If we want to access single record in the database:

* **get()**
* **Query:**

Model\_Name.objects.get(id=record no)

**Execution:**

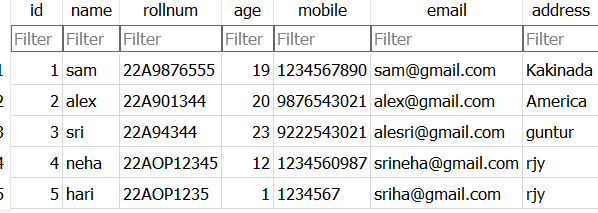
****

If you want to **access duplicates** from database

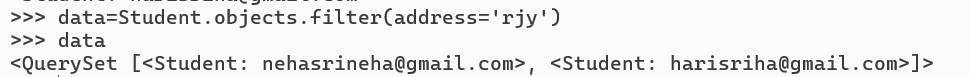
* **filter()**
* **Query:**

Model\_Name.objects.filter(column\_name=’Value’)

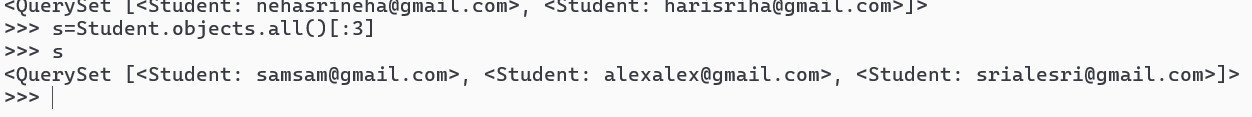
**The Table:**

****

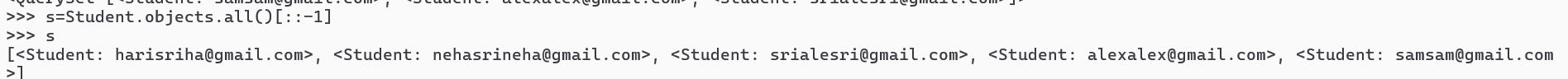
**Execution:**

****

If we want to apply the slcing to the database we use: **modelname.objects.all()[:3]**

****

**Reverse printing the data:**

****

**Alternative of the data as a output:  
How to update the particular value in the database:**

Firstly, fetch the data from the database by using record number. To update first by using get method bring the value

* **get()**

**g = Modelname.objects.get(id=2)**

We will update:

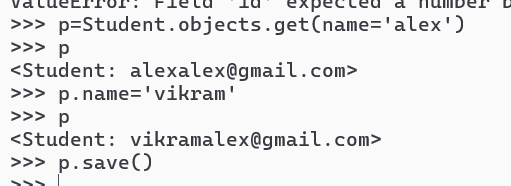
**g.column\_name = new\_VAL/ Updated value**

**g.save()**

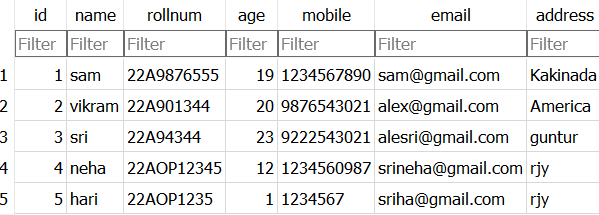
**g**

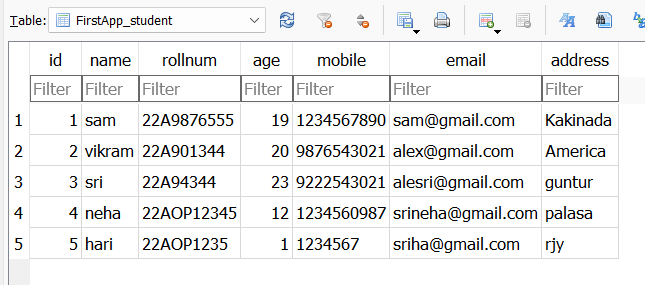
save it then the update value takes place otherwise the old value will be the output.

**Execution:**



**Update takes place in the table:**



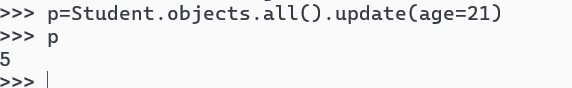
We also **update multiple values** we use all method:  
 **query:**

p= modelname.objects.all().update(email=’new value’)

p

that mean all the rows email values are updated by the new value.

**Execution:**



The table will also be updated and the cmd displays count of the rows in which the updation takes place.

**How to delete the record from the data base by using ORE queries?**

If we want to delete the record from the database then firstly we need to get the value by using get method and then delete it.

* **get()**

g=modelname.objects.get(id=2)

g

**delete:**

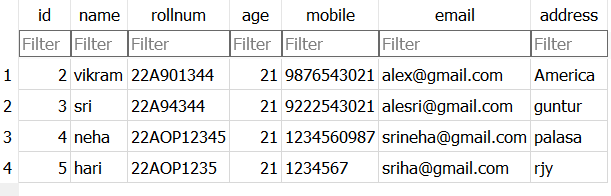
g.delete()

If a record is removed from the database then in command prompt, the removed element will be displayed in the **dictionary format**.

**Execution:**

****

**Table also updates:**

s

Now, if we need to remove all the elements from the database then we use all() method to get whole data and then delete:  
**Query:**  
 g = modelname.objects.all()

g

g.delete()

**Execution:**



And the table data will be deleted:

