

**SKILL DEVELOPMENT LAB MANUAL**

**(Node.js/React/Django)**

II YEAR CSE-Semester I

**DEPARTMENT**

**OF**

**COMPUTER SCIENCE AND ENGINEERING**

**ACADEMIC YEAR 2023-24**

|  |
| --- |
|  |

**List of Experiments**

* 1. Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid.
* 2. Make the above web application responsive web application using Bootstrap framework.
* 3. Use JavaScript for doing client – side validation of the pages implemented in experiment 1 and experiment 2.
* 4. Explore the features of ES6 like arrow functions, callbacks, promises, async/await. Implement an application for reading the weather information from openweathermap.org and display the information in the form of a graph on the web page.
* 5. Develop a java stand alone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.
* 6. Create an xml for the bookstore. Validate the same using both DTD and XSD.
* 7. Design a controller with servlet that provides the interaction with application developed inexperiment 1 and the database created in experiment 5.
* 8. Maintaining the transactional history of any user is very important. Explore the various sessiontracking mechanism (Cookies, HTTP Session)
* 9. Create a custom server using http module and explore the other modules of Node JS like OS,path, event.
* 10. Develop an express web application that can interact with REST API to perform CRUDoperations on student data. (Use Postman)
* 11. For the above application create authorized end points using JWT (JSON Web Token).
* 12. Create a react application for the student management system having registration, login, contact, about pages and implement routing to navigate through these pages.
* 13. Create a service in react that fetches the weather information from openweathermap.org and the display the current and historical weather information using graphical representation usingchart.js
* 14. Create a TODO application in react with necessary components and deploy it into github

**INTRODUCTION**

**DJANGO:**

It is a free and open source,python based web framework that follows Model-Template-Views architectural pattern.

* It is maintained by ‘’Django Software Foundation”.
* It was released on 21st July 2005.

**DJANGO IN PYTHON**:

1. Python basics
2. Functions
3. Classes-OOPS
4. Decorators
5. Generators
6. Modules
7. Packages

* It is used in ‘’Back-End Web Development Framework’’.
* It is completely developed using python.

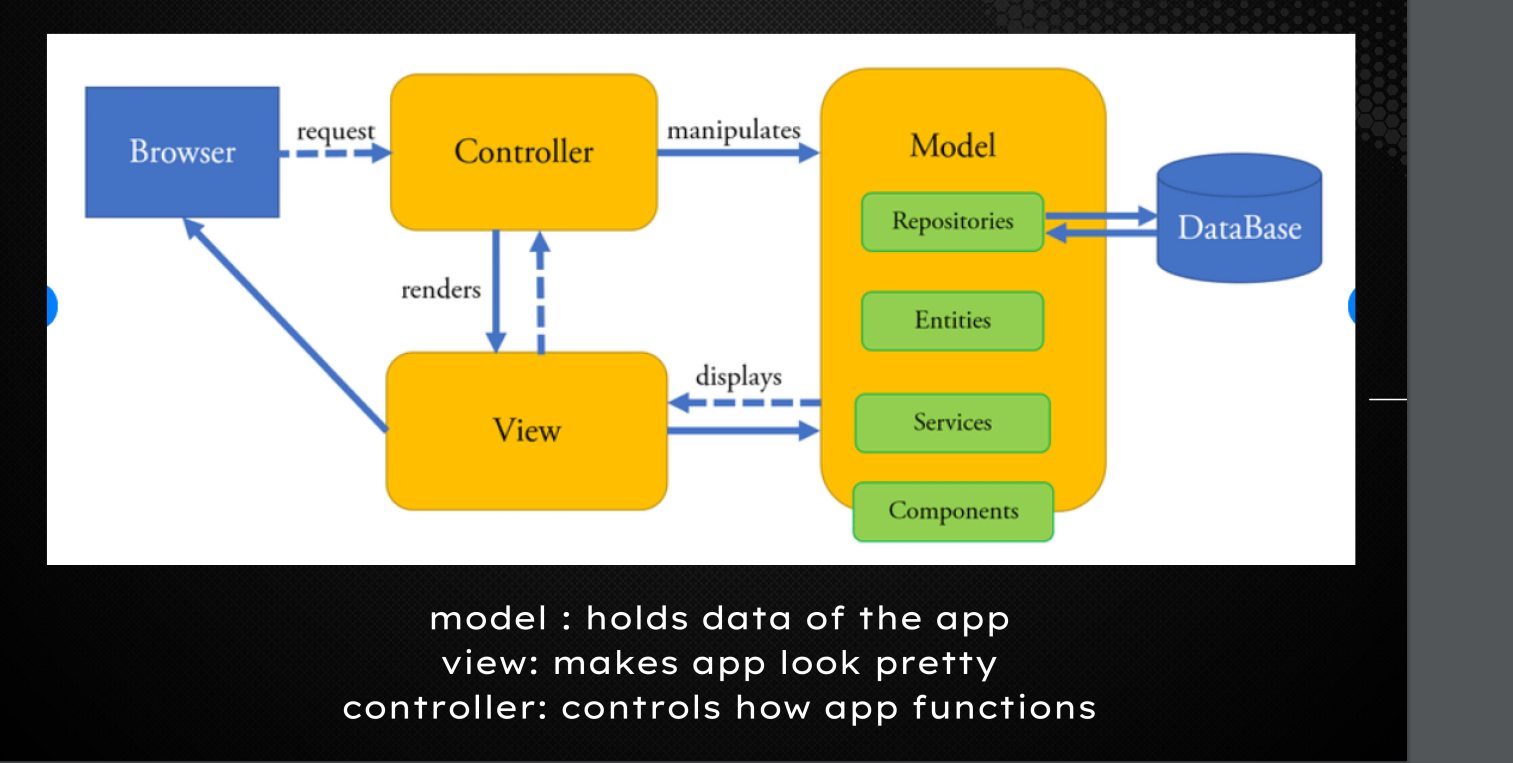
**Web Applications:**

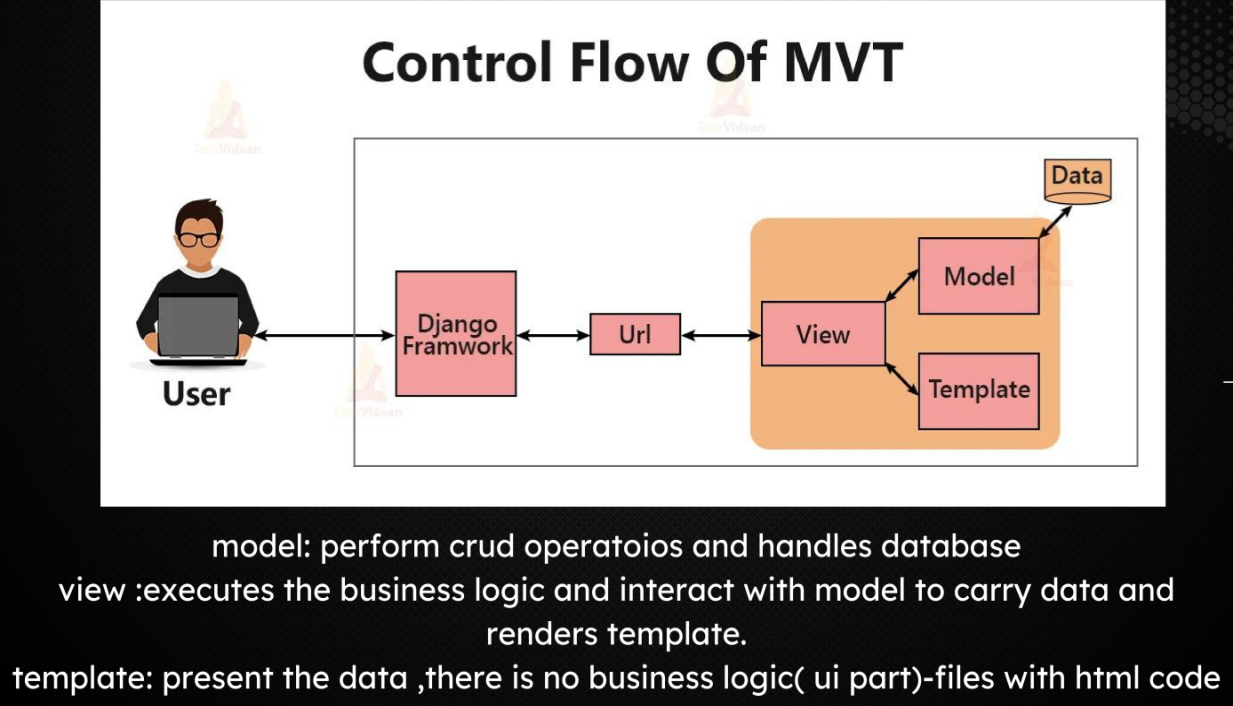
**FRONT-END:**

1. Html
2. Css
3. Javascript
4. Bootstrap
5. Jquery
6. Dom

**BACK-END**:

1. Django
2. Python
3. Databases
4. Javascript

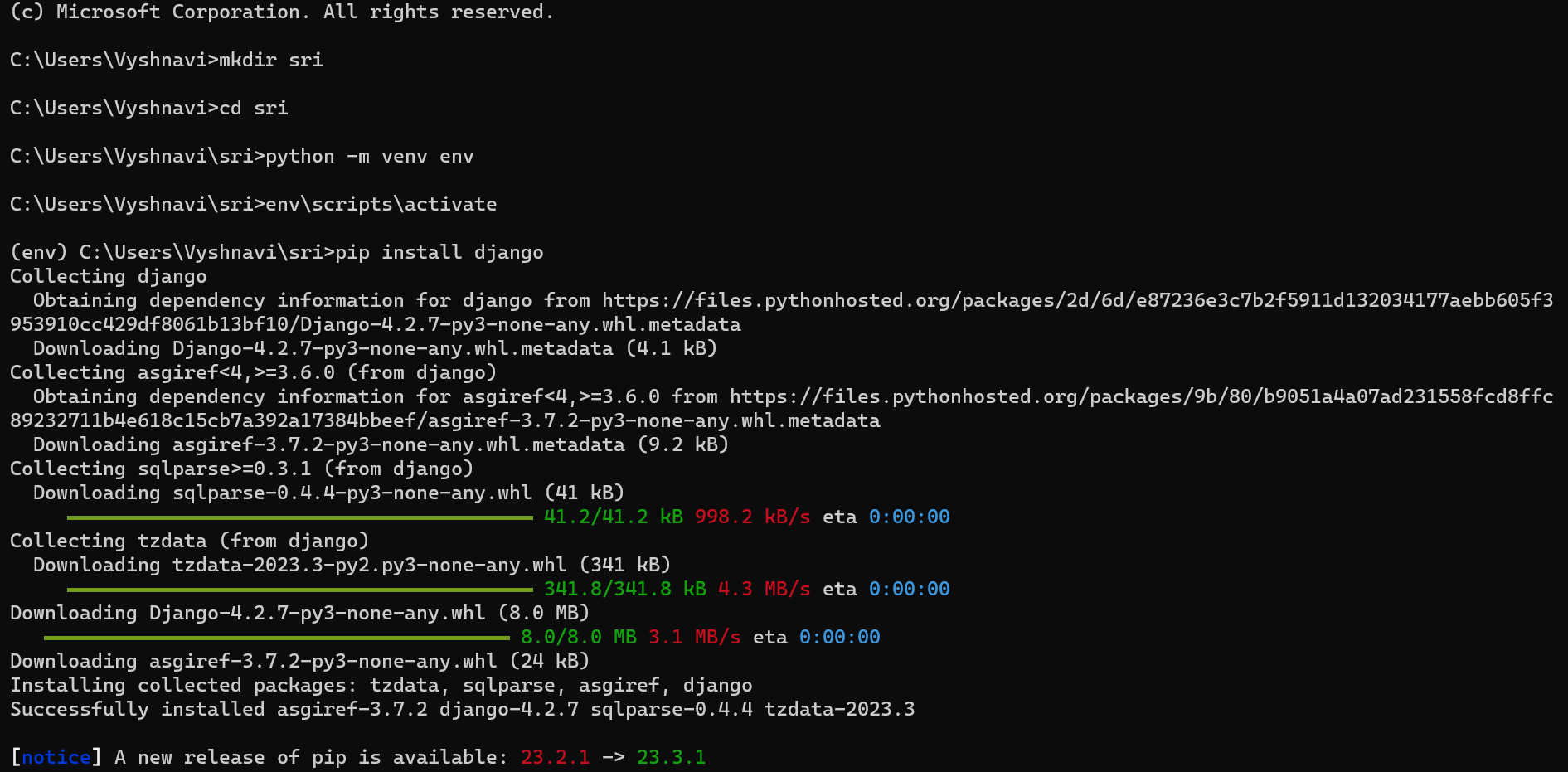






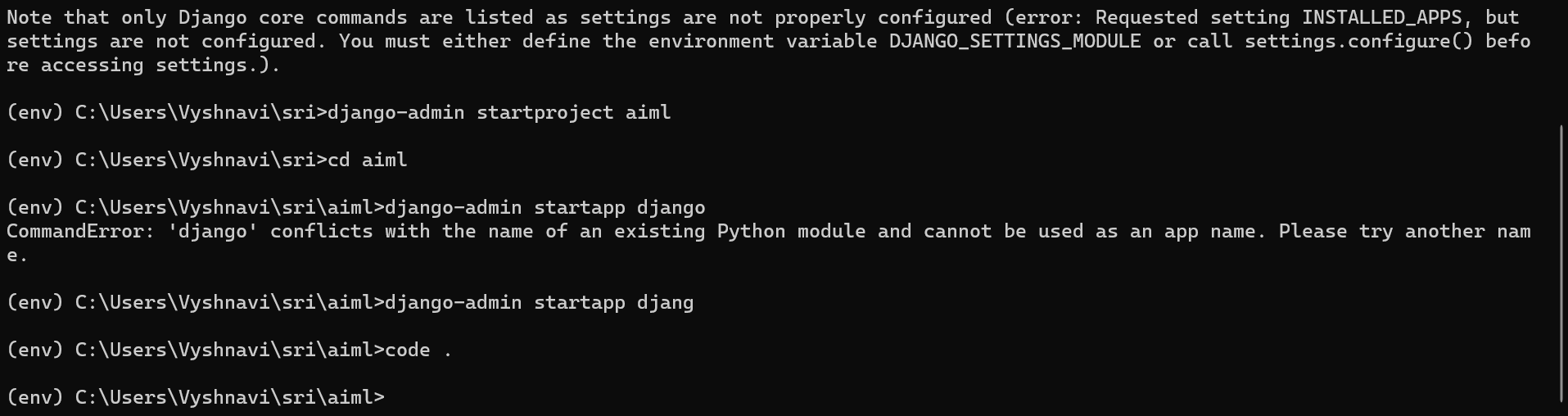








**Now VS-CODE will open..**



**EXPERIMENT-1:**

**1. Build a responsive web application for shopping cart with registration, login, catalog and cart pages using**

**CSS3 features, flex and grid.**

* This is views.py

from django.shortcuts import render,redirect

from django.contrib.auth.models import User,auth

from django.contrib import messages

# Create your views here.

def home(request):

return render(request,"home.html")

=username).exists():

messages.info(request, 'Email def register(request):

if request.method == 'POST':

first\_name = request.POST['first\_name']

last\_name = request.POST['last\_name']

username = request.POST['username']

email = request.POST['email']

password = request.POST['password']

confirm\_password = request.POST['confirm\_password']

if password==confirm\_password:

if User.objects.filter(username

is exist')

return redirect(register)

else:

user = User.objects.create\_user(username=username, password=password, email=email, first\_name=first\_name, last\_name=last\_name)

user.set\_password(password)

user.is\_staff = True

user.save()

return redirect('login\_user')

else:

print("this is not post method")

return render(request,"register.html")

def login\_user(request):

if request.method == 'POST':

username =request.POST['username']

password= request.POST['password']

user = auth.authenticate(username=username,password= password)

if user is not None:

auth.login(request, user)

return redirect('home')

else:

messages.info(request, 'Invalid Username or Password')

return redirect('login\_user')

else:

return render(request,"login.html")

def logout\_user(request):

auth.logout(request)

return redirect('home')

* This is at project level urls.py

from django.contrib import admin

from django.urls import path,include

urlpatterns = [

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

path('', include('app.urls')),

]

* This is application level urls.py

from django.contrib import admin

from django.urls import path

from .views import home

from .views import home,register,login\_user,logout\_user

urlpatterns = [

path('',home,name="home"),

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

path('login\_user',login\_user,name ="login\_user"),

path("logout\_user", logout\_user, name="logout\_user"),

]

* In templates we do have 3 files- home.html, register.html, login.html.
* **This is home. Html:**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

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

<title>authbyuser</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

</head>

<body>

<!-- As a heading -->

<nav class="navbar bg-body-tertiary">

<div class="container-fluid">

<span class="navbar-brand mb-0 h1">Navbar</span>

</div>

</nav>

<h1 class="text-center text-success">shopping cart</h1>

<div class="container-fluid">

{% if request.user.is\_authenticated %}

<p class="h3 text-success">You are Logged In</p>

<p class="text-success h3">Username : {{ request.user.username }}</p>

<a href={% url 'logout\_user' %} type="button" class="btnbtn-success btn-lg">Log Out</a>

{% else %}

<p class="text-danger" h3>User is not logged in </p>

<a href={% url 'register' %} type="button" class="btnbtn-success btn-lg">Register</a>

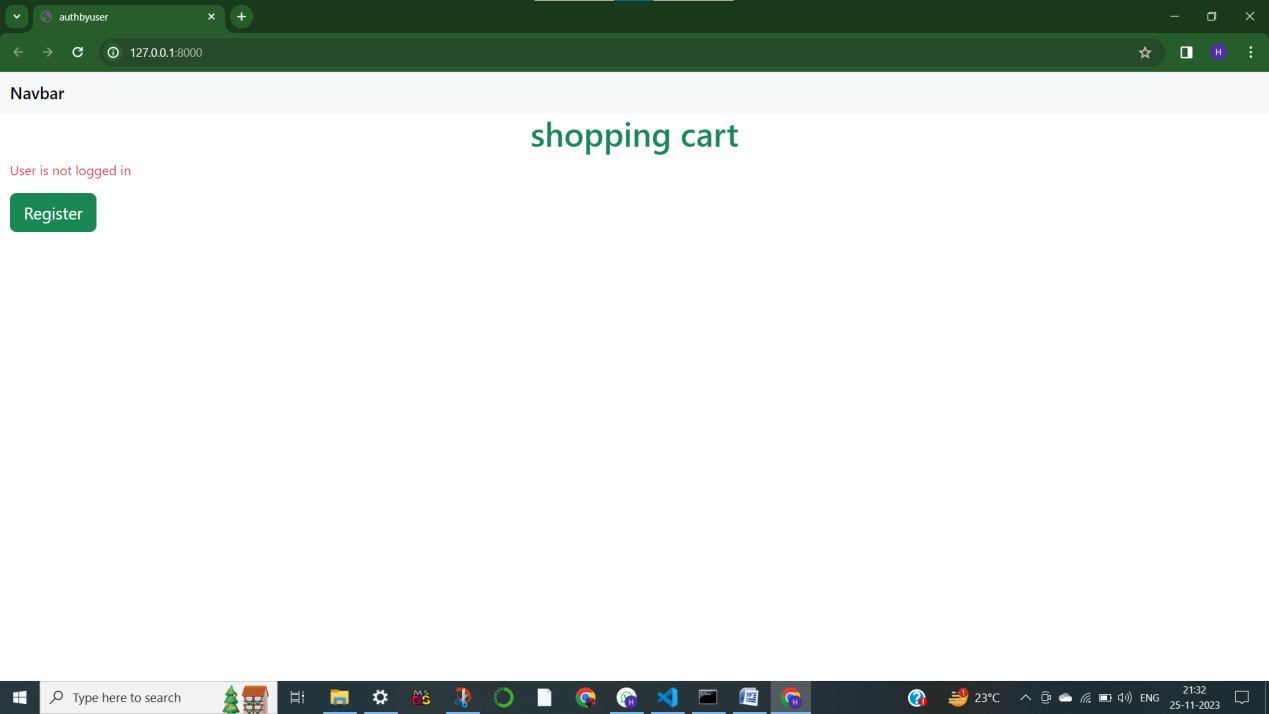
{% endif %}

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>

</body>

</html>



* **THIS IS login.html:**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

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

<title>authbyuser</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

</head>

<body>

<!-- As a heading -->

<nav class="navbar bg-body-tertiary">

<div class="container-fluid">

<span class="navbar-brand mb-0 h1">Navbar</span>

</div>

</nav>

<h1 class="text-center text-success">shopping cart-loginpage</h1>

{% if request.user.is\_authenticated %}

<p class="text-success">You are Logged In</p>

<p class="text-success">Username : {{ request.user.username }}</p>

{% else %}

<p class="text-danger" h3>User is not logged in </p>

{% endif %}

<div class="container-fluid">

<form action="{% url "login\_user"%}" method="POST">

{% csrf\_token %}

<div class="row mb-3">

<label for="inputEmail3" class="col-sm-2 col-form-label">Username</label>

<div class="col-sm-10">

<input name= "username" type="text" class="form-control" id="inputEmail3">

</div>

</div>

<div class="row mb-3">

<label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>

<div class="col-sm-10">

<input name="password" type="password" class="form-control" id="inputPassword3">

</div>

</div>

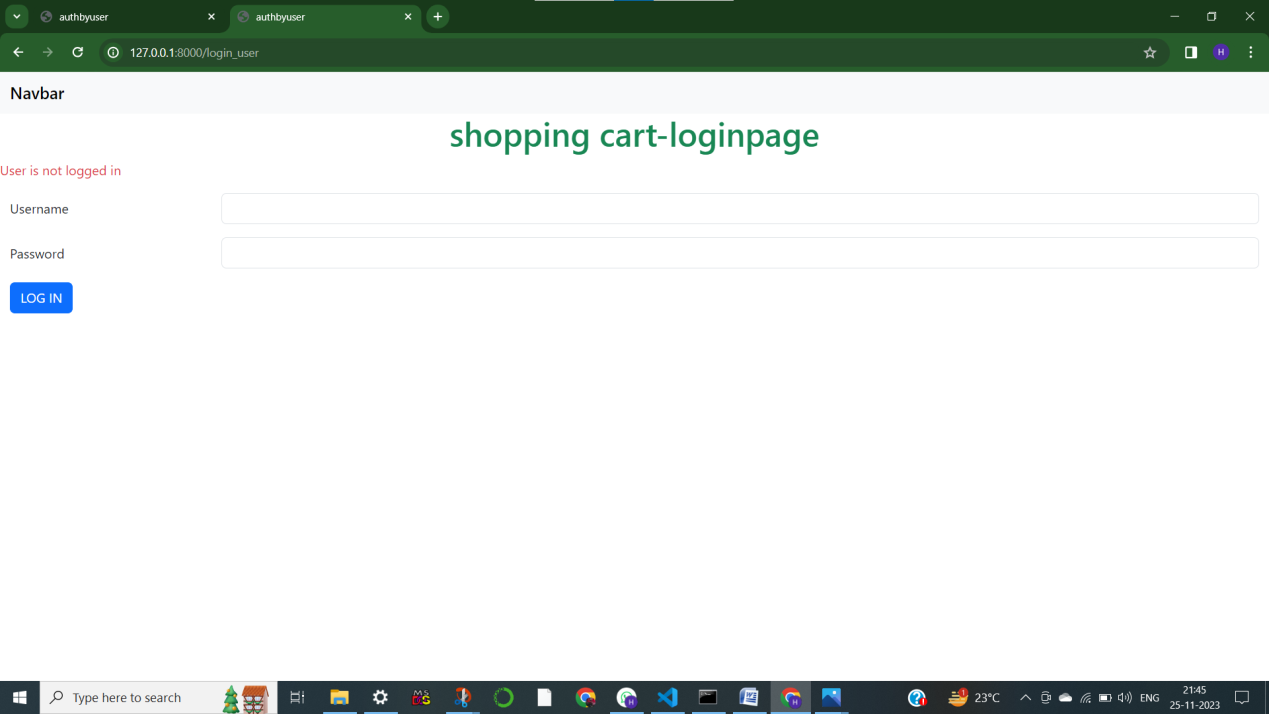
<button type="submit" class="btnbtn-primary">LOG IN</button>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>

</body>

</html>



* **This is register.html:**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

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

<title>authbyuser</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

</head>

<body>

<!-- As a heading -->

<nav class="navbar bg-body-tertiary">

<div class="container-fluid">

<span class="navbar-brand mb-0 h1">Navbar</span>

</div>

</nav>

<h1 class="text-center text-success">d-mart</h1>

<div class="container-fluid">

<form action = "{% url "register" %}" method = "POST" class="row g-3">

{% csrf\_token %}

<div class="col-md-6">

<label for="inputEmail4" class="form-label">First Name</label>

<input name = "first\_name" type="text" class="form-control" id="">

</div>

<div class="col-md-6">

<label for="inputEmail4" class="form-label">Last Name</label>

<input name = "last\_name" type="text" class="form-control" id="">

</div>

<div class="col-md-6">

<label for="inputEmail4" class="form-label">user name</label>

<input name = "username" type="text" class="form-control" id="">

</div>

<div class="col-md-6">

<label for="inputEmail4" class="form-label">Email</label>

<input name = "email" type="email" class="form-control" id="inputEmail4">

</div>

<div class="col-md-6">

<label for="inputPassword4" class="form-label">Password</label>

<input name = "password" type="password" class="form-control" id="inputPassword4">

</div>

<div class="col-md-6">

<label for="inputPassword4" class="form-label">Confirm Password</label>

<input name = "confirm\_password" type="password" class="form-control" id="inputPassword4">

</div>

<div class="col-12">

<button type="submit" class="btnbtn-success btn-lg">Register</button>

</div>

</form>

<div class="form-group">

{% for message in messages %}

<div class="alert alert-danger" role="alert">

{{message}}

</div>

{% endfor %}

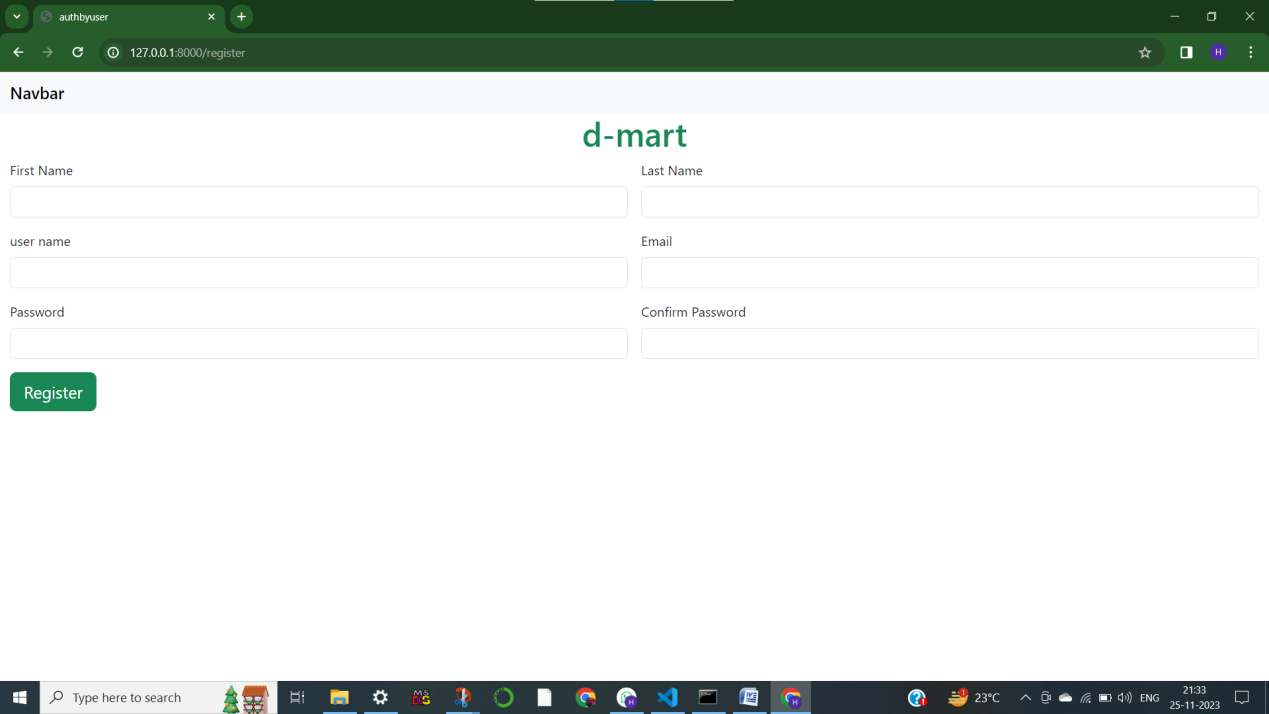
</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>

</body>

</html>



**VIVA QUESTIONS**

1. What is Django?How does Django works?
2. Who invented Django.
3. Give latest version of Django
4. Write the command to check if your system has PIP installed.
5. Write the command to check version number of Django.
6. What is an App?
7. Explain Django Views.
8. Explain in brief about below tags

* <h1>
* <p>
* <a>
* <img>
* <div>

1. What is CSS?
2. Write a short note on following form elements:

* <input>
* <label>
* <select>
* <textarea>
* <button>
* <fieldset>
* <legend>
* <datalist>
* <output>
* <option>
* <optgroup>

**EXPERIMENT 2**

**Make the above web application responsive web application using Bootstrap framework.**

***WEB APPLICATIONS***

***USING BOOTSTRAPFRAMEWORK.***

INTRODUCTION:

**What is Bootstrap?**

Bootstrap is an open-source toolkit created by Twitter developers to assist in building web applications and websites. It consists of HTML, CSS, and JavaScript components and utilities, providing a cohesive set of tools for creating consistent and visually appealing user interfaces.

**Features of Bootstrap:**

1. **Responsive Grid System:** Bootstrap utilizes a responsive, mobile-first grid system that helps create layouts that adapt to various screen sizes and devices.
2. **Pre-styled Components:** It offers a library of components such as buttons, forms, navigation bars, dropdowns, modals, and more, all styled and ready to use.
3. **Customizable Styles:** Bootstrap's styles are easily customizable. You can modify colors, typography, spacing, and other elements to match your project’s design requirements.
4. **JavaScript Plugins:** Bootstrap includes several JavaScript plugins, like carousels, modals, tooltips, and more, enhancing interactivity and functionality without the need for additional code.

This is views.py

fromdjango.shortcutsimportrender

defhome(request):

    returnrender(request,"home.html")

this is project level urls.py

#from django.contrib import admin

fromdjango.urlsimportpath

urlpatterns= [

   # path('admin/', admin.site.urls),

]

This is app level urls.py

fromdjango.contribimportadmin

fromdjango.urlsimportpath,include

urlpatterns= [

    path('admin/', admin.site.urls),

    path('',include('app.urls'))

]

This is in home.html

<!doctypehtml>

<htmllang="en">

  <head>

    <metacharset="utf-8">

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

    <title>website</title>

    <linkhref="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"rel="stylesheet"integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"crossorigin="anonymous">

  </head>

  <body>

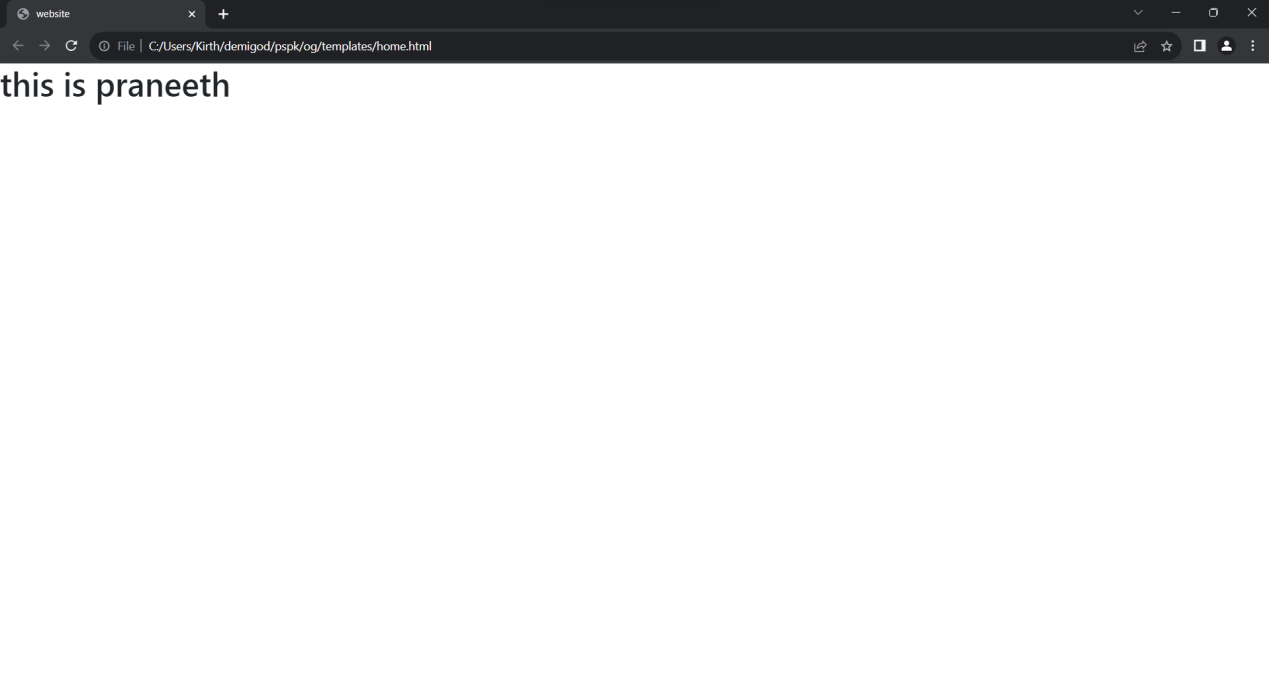
    <h1>this is praneeth</h1>

    <scriptsrc="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"crossorigin="anonymous"></script>

  </body>

</html>

Output:



**VIVA QUESTIONS**

1. What is Bootstrap?
2. What is Responsive Web design?
3. Write advantages of bootstrap.
4. How to create web page with Bootstrap.
5. Explain the following statement

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

1. Write Bootstrap 5 VS Bootstrap 3&4
2. Explain the following statements
   * <meta charset="utf-8">
   * <meta name="viewport" content="width=device-width, initial-scale=1">
3. Write features of Bootstrap.
4. Write a short note on .table-responsive class .
5. Write a short note on .tableclass .

**Experiment 3:**

**Use JavaScript for doing client –side validation of the pages implemented in experiment 1 and experiment 2.**

**Experiment 4:**

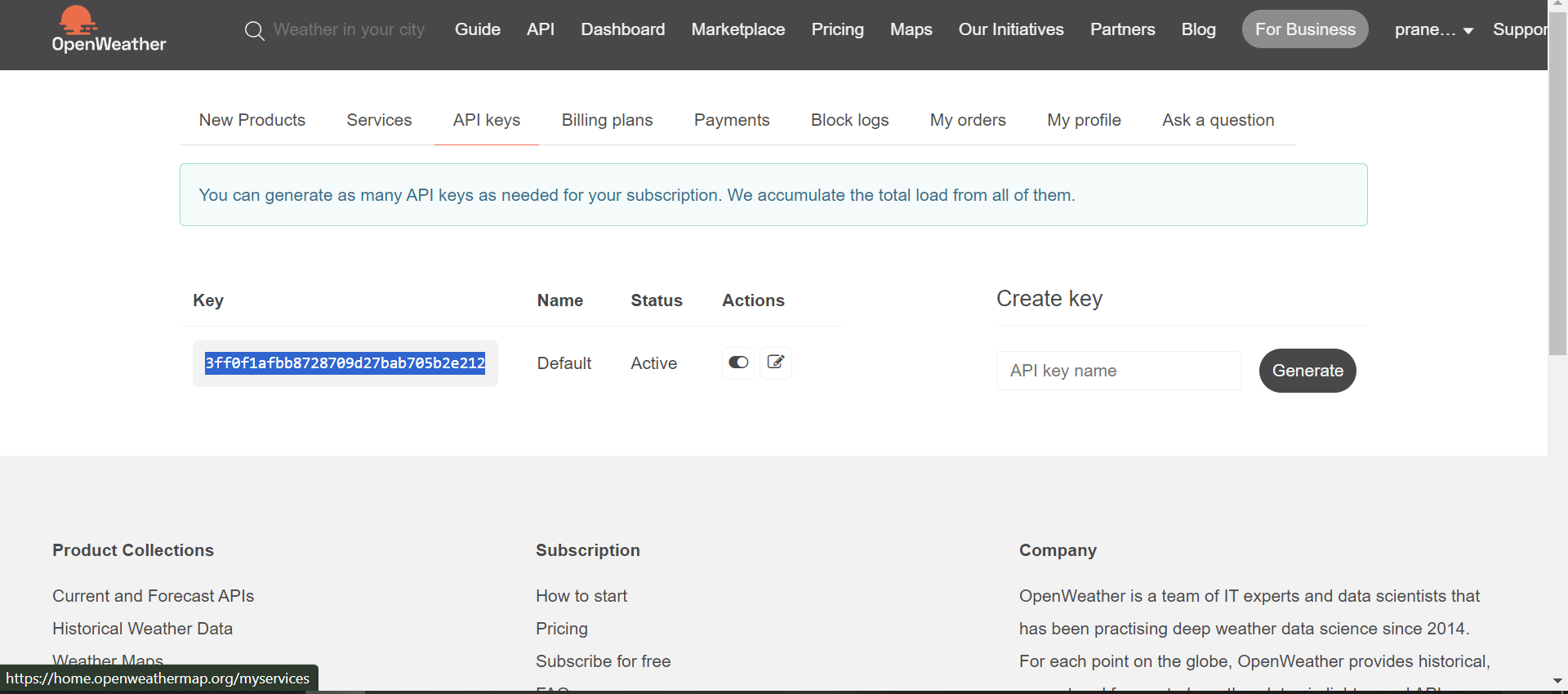
**Explore the features of ES6 like arrow functions, call backs, promises, async/wait. Implement an application for reading the weather information from open weathermap.org and display the information in the form of a graph on the web page.**

**WEATHER APP**

**STEPS TO CREATE WEATHER PROJECT** .

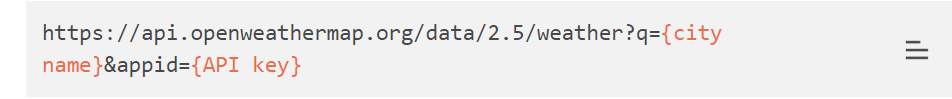
1. CREATE THE PROJECT AND APPLICATION USING CMD(DJANGO INSTALLATION PROCESS) .
2. PROJECT NAME – WEATHERPROJECT.
3. APPLICATION NAME – WEATHERAPP.
4. CREATE THE APPLICATION LEVEL URLS.PY

**OPEN WEATHER WEBSITE AND SIGN IN.**



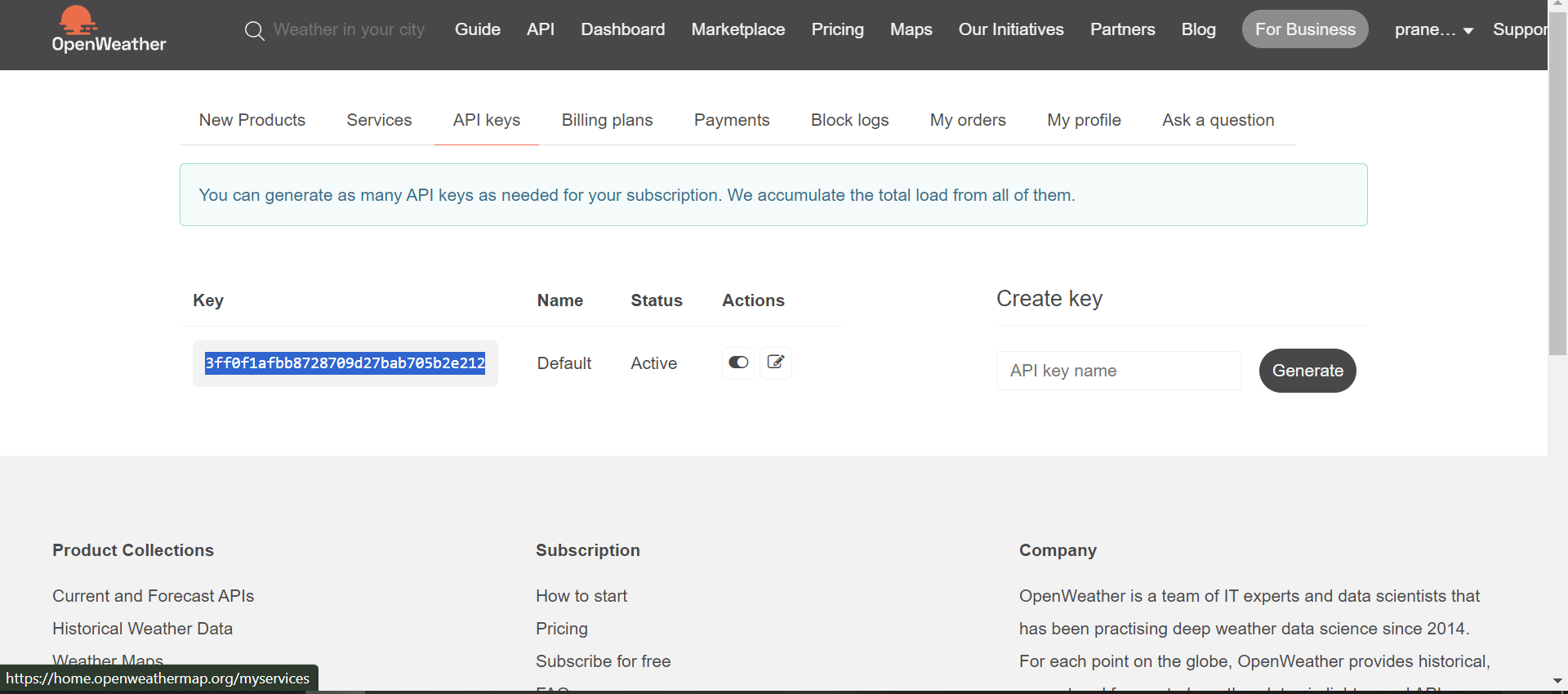
CLICK ON API, THEN CLICK ON THE API DOC UNDER THE CONTINUE WEATHER REPORT .

COPY THE URL IN THE NEW WEB PAGE .

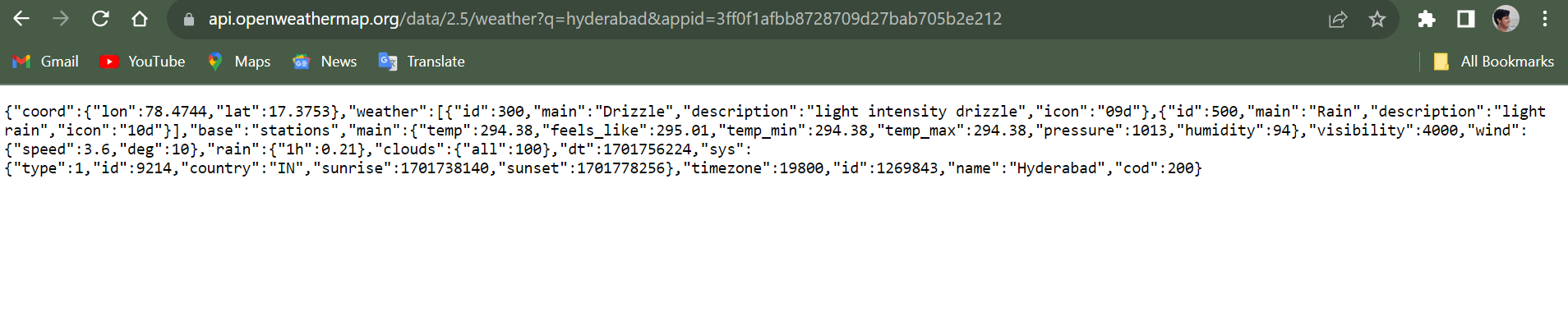


CITY NAME – REQUIRED

API KEY – PROFILE KEY .



AS PER HIGHLIGHTED FORMAT



NOW IN VS CODE

IN VIEWS.PY:

fromdjango.shortcutsimportrender

# import json to load json data to python dictionary

importjson

# urllib.request to make a request to api

importurllib.request

defindex(request):

    ifrequest.method=='POST':

        city=request.POST['city']

        ''' api key might be expired use your own api\_key

            place api\_key in place of appid ="your\_api\_key\_here " '''

        # source contain JSON data from API

        source=urllib.request.urlopen('https://api.openweathermap.org/data/2.5/weather?q='+city+'&appid=f39864907892c57a3ac10b3b4c73db7d').read()

        # converting JSON data to a dictionary

        list\_of\_data=json.loads(source)

        # data for variable list\_of\_data

        data= {

            "country\_code": str(list\_of\_data['sys']['country']),

            "coordinate": str(list\_of\_data['coord']['lon']) +' '

                        +str(list\_of\_data['coord']['lat']),

            "temp": str(list\_of\_data['main']['temp']) +'k',

            "pressure": str(list\_of\_data['main']['pressure']),

            "humidity": str(list\_of\_data['main']['humidity']),

        }

        print(data)

    else:

        data={}

    returnrender(request, "index.html",data)

NOW IN APPLICATION URLS.PY :

fromdjango.urlsimportpath

from .importviews

urlpatterns=[  path('', views.index),

]

NOW IN PROJECT LEVEL URLS.PY :

fromdjango.contribimportadmin

fromdjango.urlsimportpath, include

urlpatterns= [

    path('admin/', admin.site.urls),

    path('',include('weatherapp.urls')),

]

NOW IN INDEX.HTML

<!DOCTYPEhtml>

<htmllang="en"dir="ltr">

<head>

  <metacharset="utf-8">

  <title>weather</title>

  <!-- Latest compiled and minified CSS -->

  <linkrel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">

  <!-- jQuery library -->

  <scriptsrc="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

  <!-- Latest compiled JavaScript -->

  <scriptsrc="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

</head>

<body>

  <navclass="row"style="background: red; color: white;">

    <h1class="col-md-3 text-center">weather</h1>

  </nav>

  <br/>

  <br/>

  <centerclass="row">

    <formmethod="post"class="col-md-6 col-md-offset-3">

      {% csrf\_token %}

      <divclass="input-group">

        <inputtype="text"class="form-control"name="city"placeholder="Search">

        <divclass="input-group-btn">

          <buttonclass="btnbtn-default"type="submit">

            <iclass="glyphiconglyphicon-search"></i>

          </button>

        </div>

        <form>

  </center>

  <divclass="row">

    {% if country\_code and coordinate and temp and pressure and humidity %}

    <divclass="col-md-6 col-md-offset-3">

      <h3>country code : {{country\_code}}</h1>

        <h5>coordinate : {{coordinate}}</h5>

        <h5>temp : {{temp}}</h5>

        <h5>pressure : {{pressure}} </h5>

        <h5>humidity : {{humidity}}</h5>

    </div>

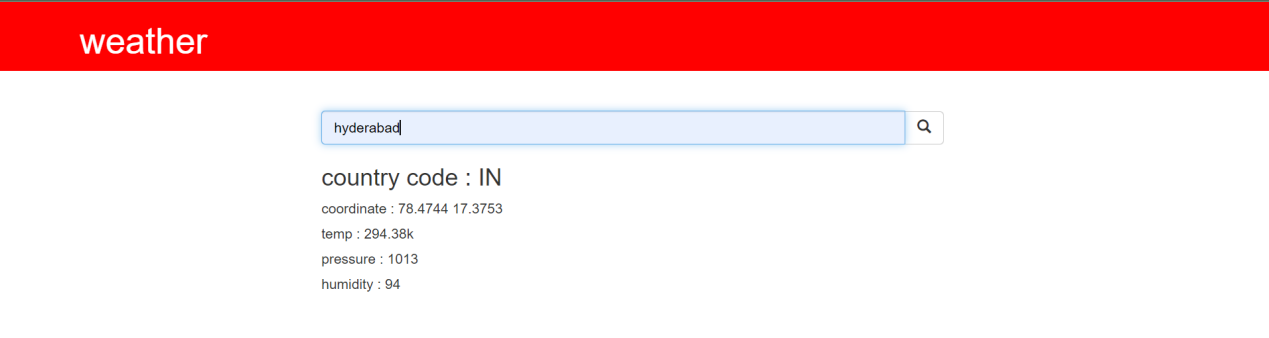
    {% endif %}

  </div>

</body>

</html>

OUTPUT :



**VIVA QUESTIONS**

1. Write steps to create app.
2. Write short notes on Django Views.
3. What is the function of path()?
4. Give use of urlpatterns[] list.
5. Define render() function.
6. Explain about json package.
7. Explain about the below statement

urlpatterns= [

    path('admin/', admin.site.urls),

    path('',include('weatherapp.urls')),

]

1. How to import python libraries in Django.
2. What happens when we write request.method == "GET"

Or request.method == "POST"

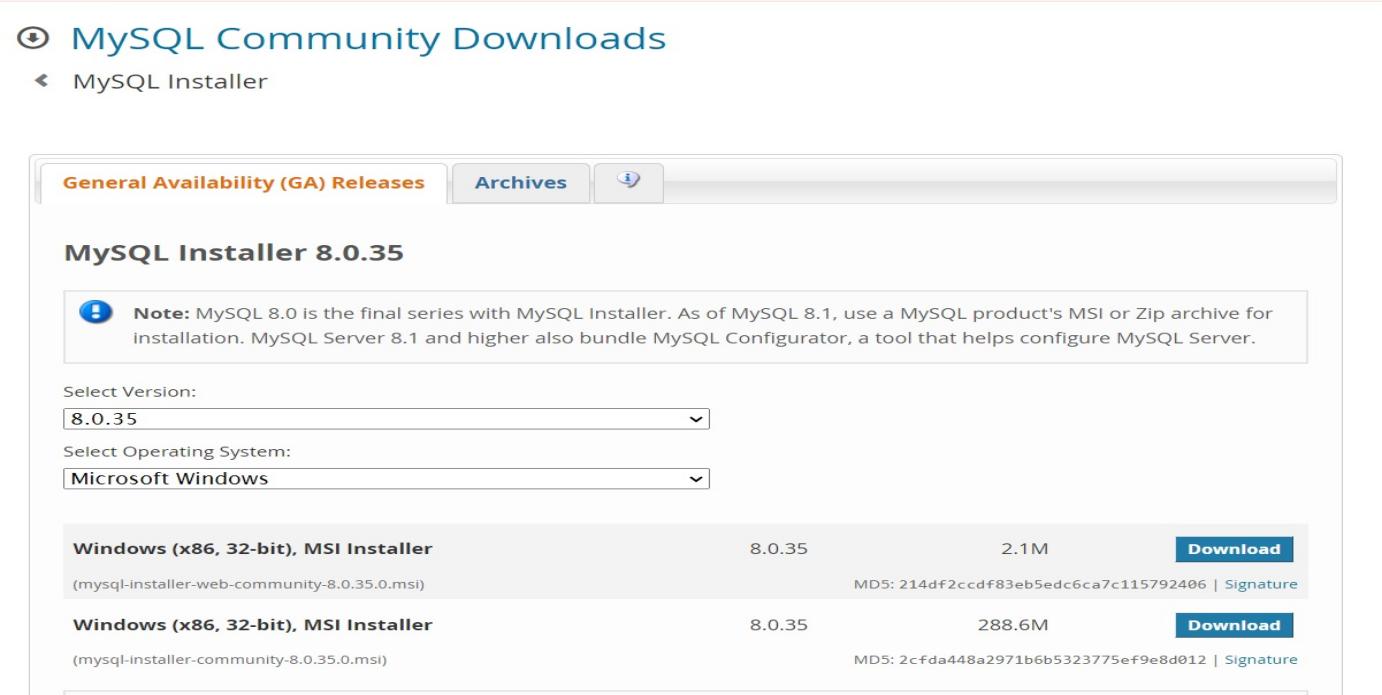
1. How to fetch internet resources.

**Experiment 5:**

**Develop a java stand-alone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.**

1) go through mysqlinstaller.

dev.mysql.com/downloads/installer/



2) download

mysql server-inside mysql server

mysql shell - inside application

3)create a password root;

4)pip install mysqlclient in terminal of vs code and in cmd of system.

5)open mysqlcommandline client and after entering password, sql will be activated

---> create database databasename;

this created database should be called in settings of django

--->

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'praneeth',--> database name

'USER': 'root',

'PASSWORD': 'root',

'PORT': '3306',

'HOST': 'localhost',

}

}

6) check with the server

python manage.py runserver

7) django should be linked with mysql

by applying python manage.py migrate

8)in sqlcommand line client

--->use database name

---->show tables;

1. this will show you the tables of Django in the MySQL.

Steps in creating database in MySQL client

1) Create Database database name;

2) Use database name;

3) Show tables;

4) Python manage.py migrate

1. Show tables;

This will reflect the connection with django

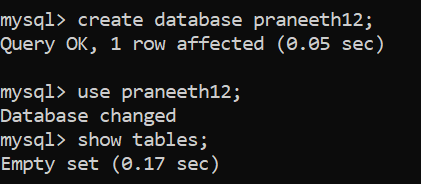
Show databases ;- (all capital) for displaying created db tables

Crud operations in databases:

-------------------------------------------------

1) Create database database name;

1. Use database name;



3)Show tables:- for displaying information inside database

4) SHOW DATABASES; - for displaying already created databases

5) Use database name: - SHOW TABLES; for usage and displaying tables

6) Create table

CREATE TABLE students (std\_id INT PRIMARY KEY, std\_name VARCHAR (240) NOT NULL, std\_program VARCHAR (240) NOT NULL, std\_stream VARCHAR (100) NOT NULL);

7) SHOW TABLES:- for displaying the newly created table in this .

EXPLAIN STUDENTS; - for newly created lists information of students table

9) Inserting into the student table

INSERT INTO students(std\_id, std\_name, std\_program, std\_stream) VALUES(1, 'praneeth', 'four years', 'computerscience');

10) SELECT \* from students; - to view all the tables in the students table

11) To get to view selected table row

SELECT \* from students WHERE std\_id = 1;

12) To update any thing in the table

UPDATE students SET std\_stream = "civil" WHERE std\_id = 2;

13)to view the updated student table

SELECT \* from students;

14)if want to delete the updated rows in the student table

DELETE FROM students WHERE std\_id = 2;

15))to view the updated student table

SELECT \* from students;

16) to drop all the table

DROP TABLE students;

17)to show the tables

SHOW tables;

In settings.py add database to the code

**Viva questions**

1.Explain below statement

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'praneeth',--> database name

'USER': 'root',

'PASSWORD': 'root',

'PORT': '3306',

'HOST': 'localhost',

}

2.How to link Django with mySQL

3.What are CURD operations.

4.What is Show tables command.

5.Write syntax for create command

6.Write Syntax for insert command.

7.Write short note on select and delete command.

8.Explain UPDATE students SET std\_stream = "civil" WHERE std\_id = 2;

9.Define Primary Key.

10.Define Database**.**

**Experiment 6:**

**Create an Xml for the bookstore, Validate the same using both DTD and XSD.**

**Bookstore.xml**

<!DOCTYPE bookstore [

<!ELEMENT bookstore (book+)>

<!ELEMENT book (title, author, price)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ELEMENT price (#PCDATA)>

]>

<bookstore xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="bookstore.xsd">

<book>

<title>Book 1</title>

<author>Author 1</author>

<price>20.00</price>

</book>

<book>

<title>Book 2</title>

<author>Author 2</author>

<price>25.00</price>

</book>

<!-- Add more books as needed -->

</bookstore>

**Bookstore.xsd**

<!-- bookstore.xsd -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="bookstore">

<xs:complexType>

<xs:sequence>

<xs:element name="book" maxOccurs="unbounded">

<xs:complexType>

<xs:sequence>

<xs:element name="title" type="xs:string"/>

<xs:element name="author" type="xs:string"/>

<xs:element name="price" type="xs:decimal"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output**:

**Viva Questions:**

1. What is Xml?
2. Difference between Xml and Html.
3. Difference between DTD and XSD.
4. Define Schema.
5. What is a well formed XML document?

**Experiment 7:**

**Design a controller with servlet that provides the interaction with application developed in experiment 1 and the database created in experiment 5.**

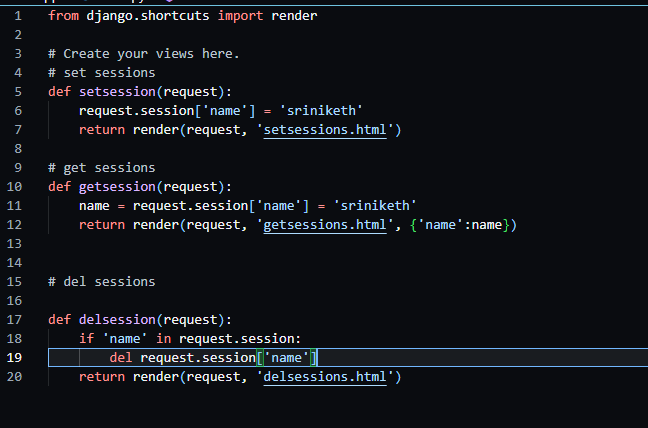
**Experiment 8:**

**Maintaining the transactional history of any user is very important .Explore the various session tracking mechanism (cookies ,HTTP session)**

**Steps:**

1. **Create a** Django project named as sessionapp.
2. **Create a** Django app named as seapp

**This is views.py:**

****

**This is project level urls.py:**

**from django.contrib import admin**

**from django.urls import path**

**from seapp import views**

**urlpatterns = [**

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

**path('set/', views.setsession),**

**path('get/', views.getsession),**

**path('del/', views.delsession),**

**]**

No need to create app level urls.py..

3]create templates folder in app level.

**delsessions.html:**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<meta name="viewport" content = "width=device-width,initial-scale=1.0">

<title>del sessions</title>

</head>

<body>

<h4>sessions deleted</h4>

</body>

</html>

**setsessions.html:**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<meta name="viewport" content = "width=device-width,initial-scale=1.0">

<title>set sessions</title>

</head>

<body>

<h4>set sessions</h4>

{{name}}

</body>

</html>

**getsessions.html:**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<meta name="viewport" content = "width=device-width,initial-scale=1.0">

<title>get sessions</title>

</head>

<body>

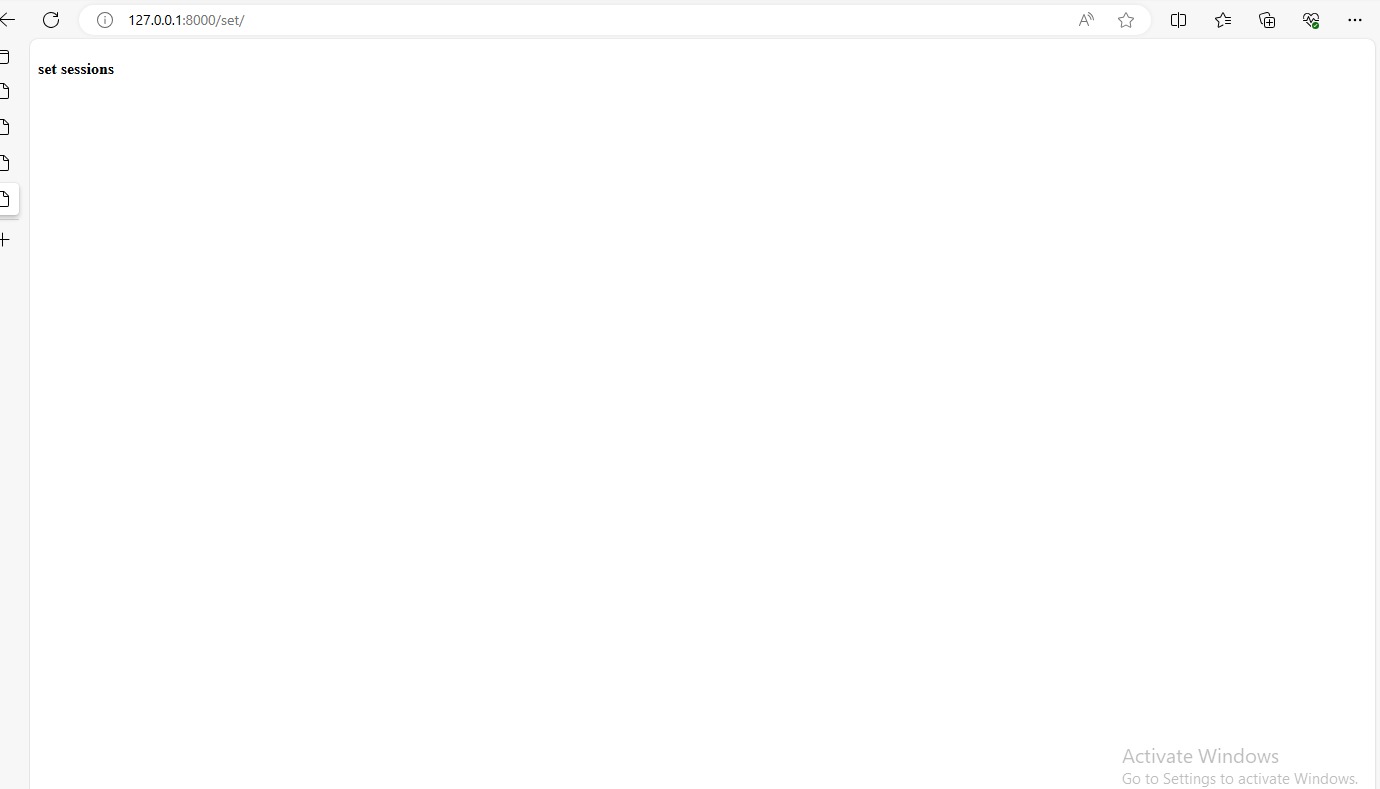
<h4>get sessions</h4>

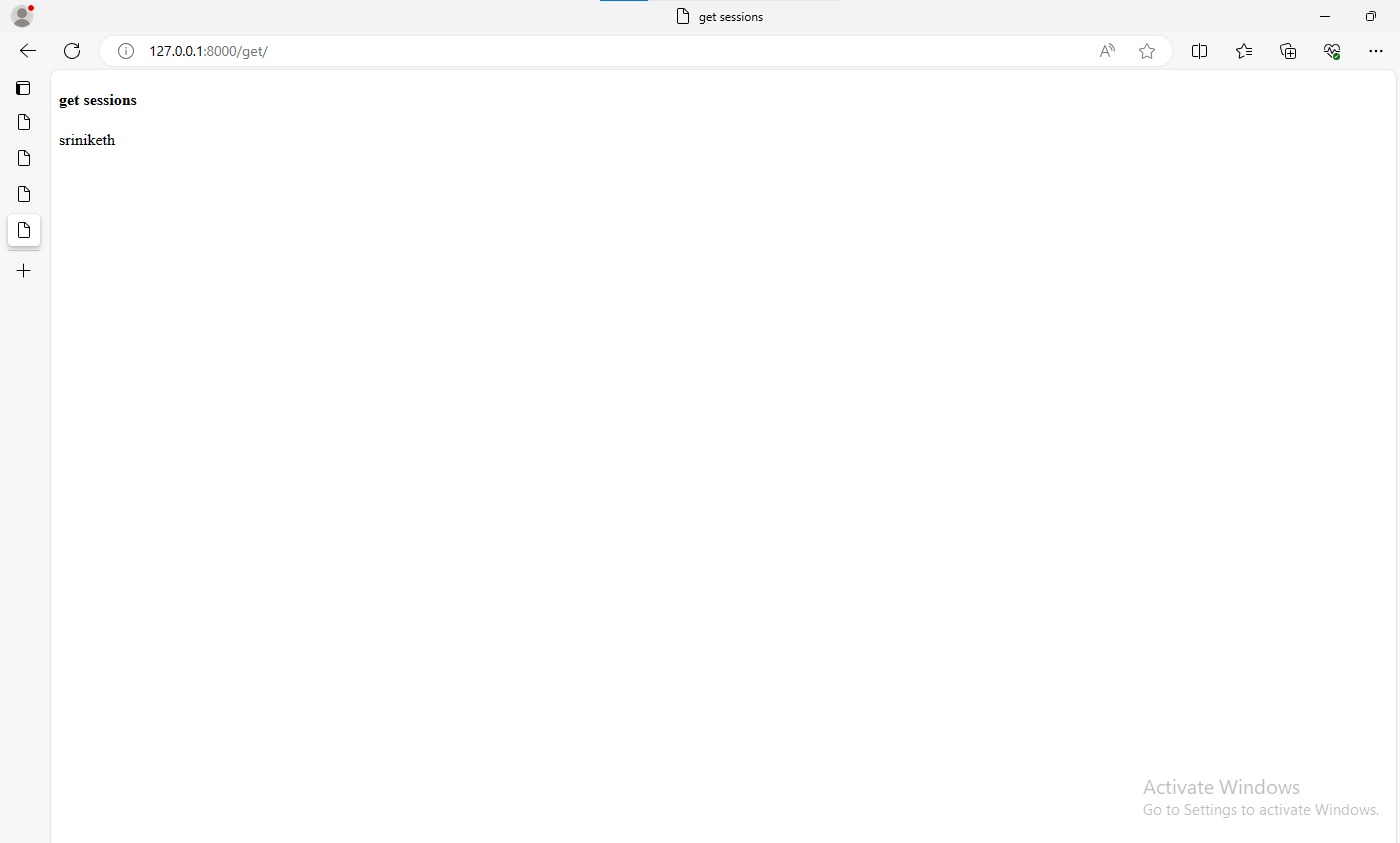
{{name}}

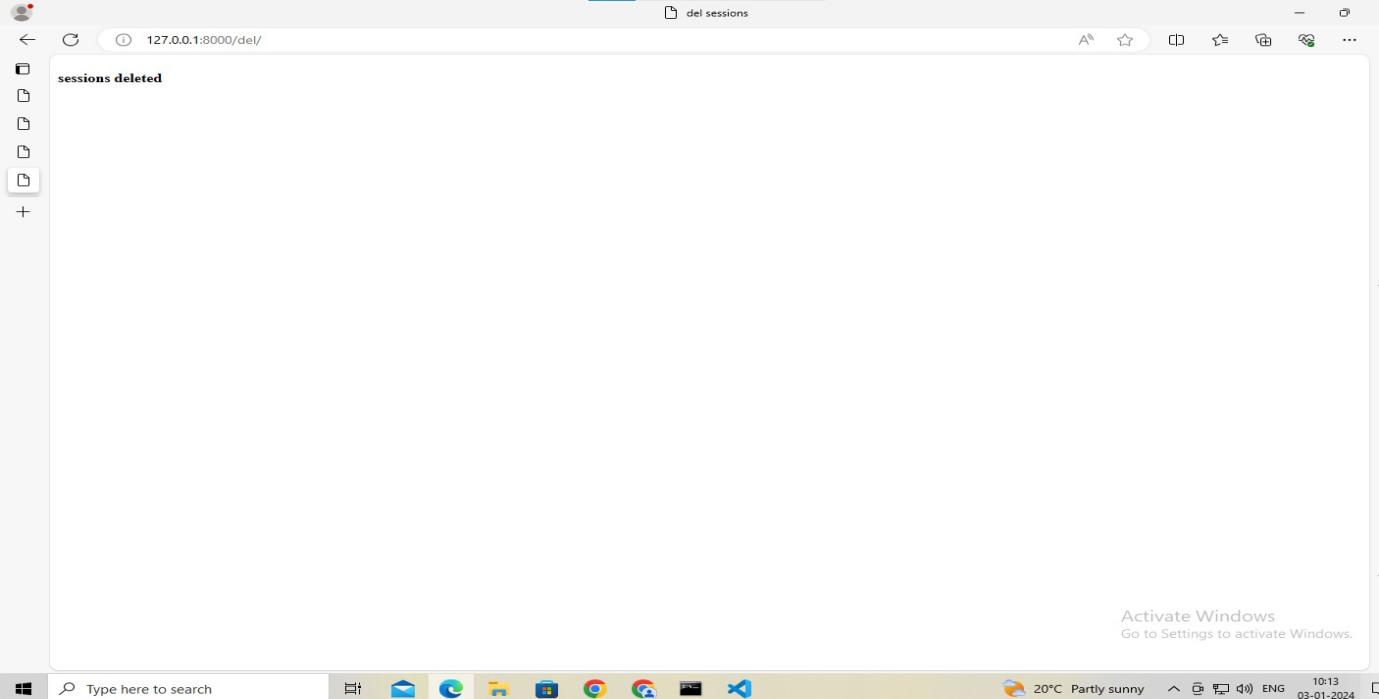
</body>

</html>

**OUTPUT:**

****

****

****

**Cookies:**

1. **Create a project**
2. **Create a app in the command prompt**

**This is views.py:**

from django.shortcuts import render

from datetime import datetime, timedelta

# Create your views here.

# set sessions

def setcookie(request):

    response =  render(request, 'cookieapp/setcookies.html')

    response.set\_cookie('name', 'praneeth', expires = datetime.utcnow()+timedelta(days=2))

    return response

# get sessions

def getcookie(request):

    name = request.COOKIES.get('name')

    return render(request, 'cookieapp/getcookies.html', {'name':name})

# del sessions

def delcookie(request):

    response =  render(request, 'cookieapp/delcookies.html')

    response.delete\_cookie('name')

    return response

**This is project level urls.py:**

from django.contrib import admin

from django.urls import path

from cookieapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('set/', views.setcookie),

    path('get/', views.getcookie),

    path('del/', views.delcookie),

]

**THIS IS getcookie.html :**

**<!DOCTYPE html>**

**<html lang = "en">**

**<head>**

**<meta charset = "UTF-8">**

**<meta name="viewport" content = "width=device-width,initial-scale=1.0">**

**<title>get cookie</title>**

**</head>**

**<body>**

**<h4>get cookie</h4>**

**{{name}}**

**</body>**

**</html>**

**This is setcookie.html:**

**<!DOCTYPE html>**

**<html lang = "en">**

**<head>**

**<meta charset = "UTF-8">**

**<meta name="viewport" content = "width=device-width,initial-scale=1.0">**

**<title>set cookie</title>**

**</head>**

**<body>**

**<h4>set cookie</h4>**

**{{name}}**

**</body>**

**</html>**

**This is delcookie.html :**

**<!DOCTYPE html>**

**<html lang = "en">**

**<head>**

**<meta charset = "UTF-8">**

**<meta name="viewport" content = "width=device-width,initial-scale=1.0">**

**<title>del cookies</title>**

**</head>**

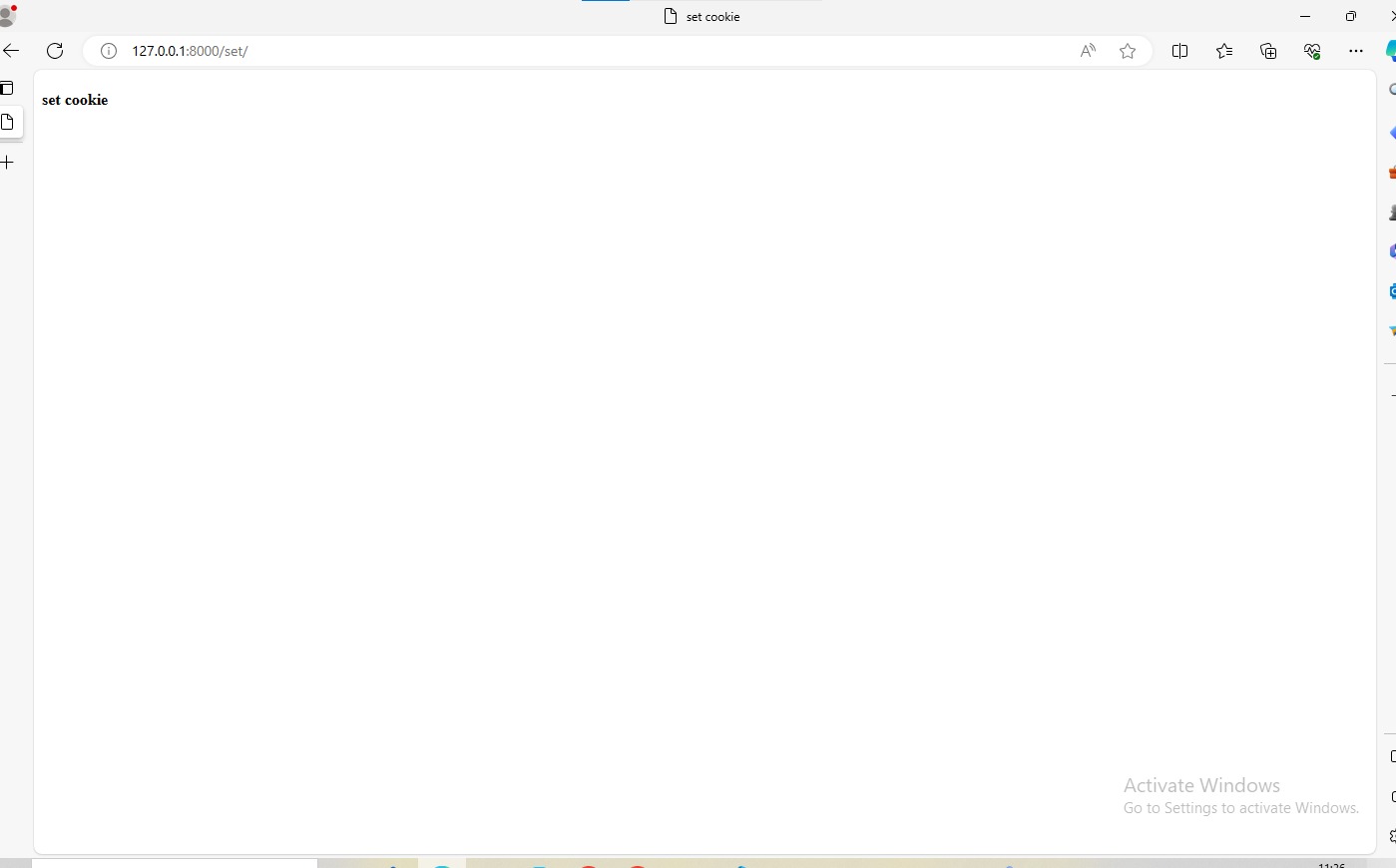
**<body>**

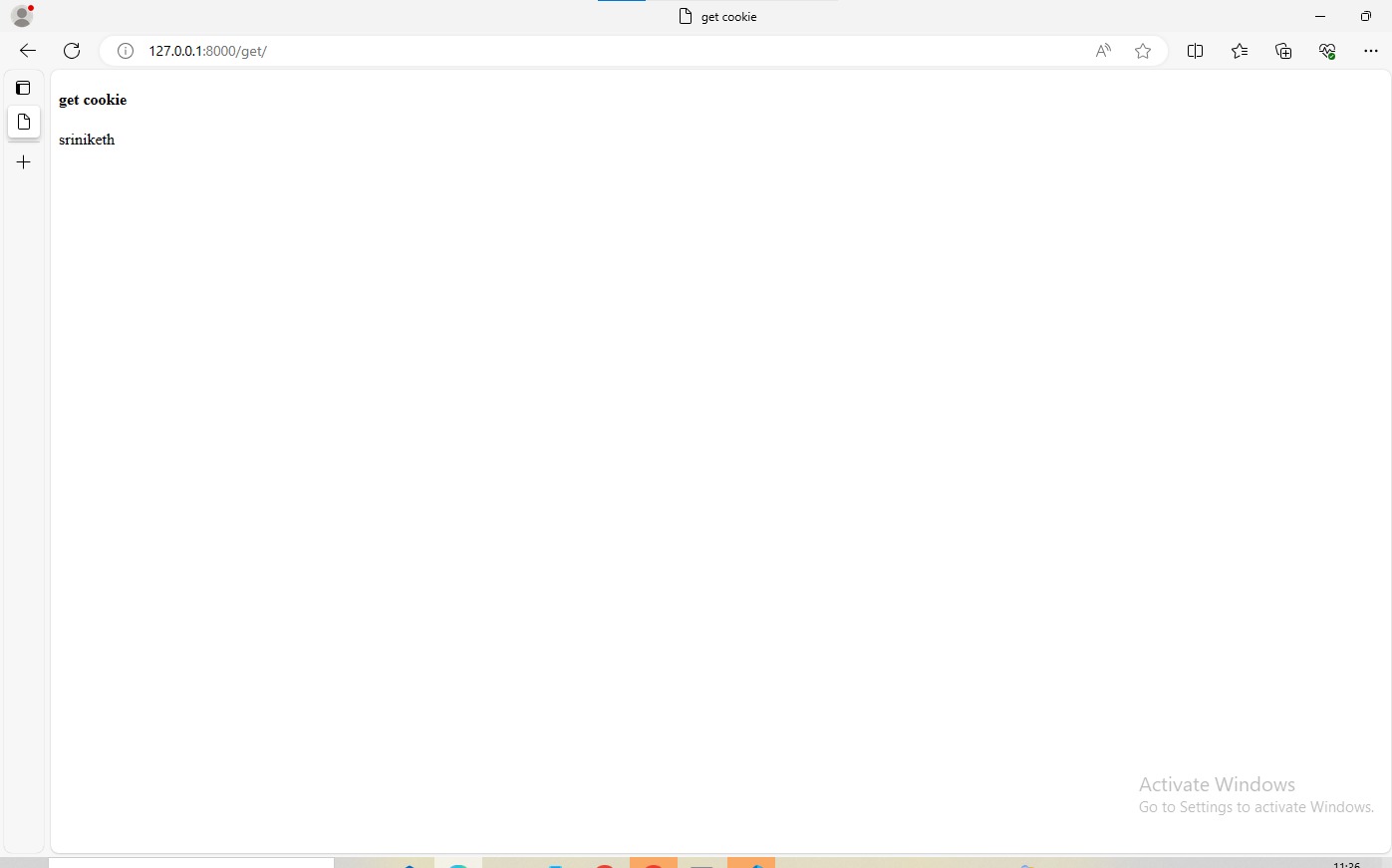
**<h4>cookies deleted</h4>**

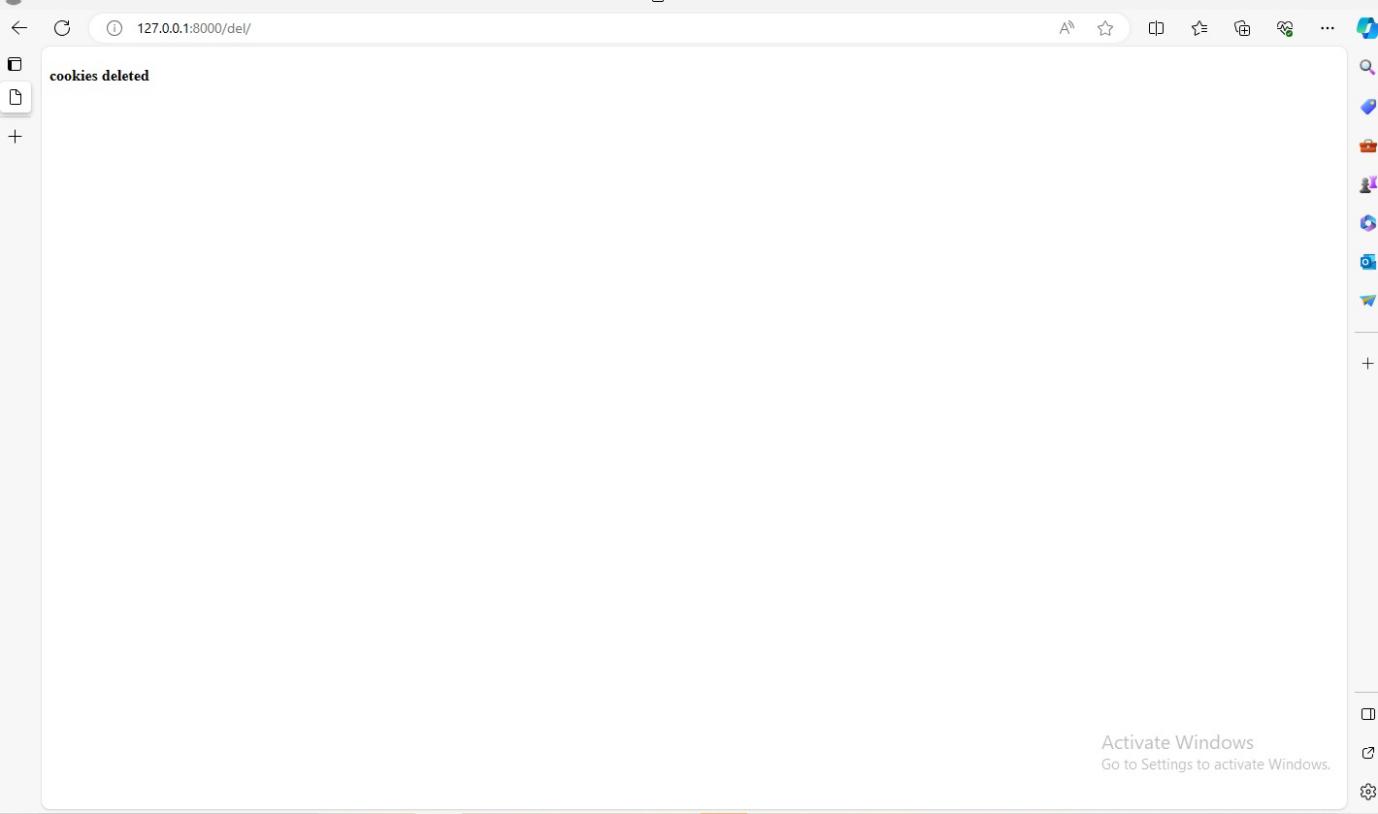
**</body>**

**</html>**

**Outputs:**

****

****

****

**Viva Questions**

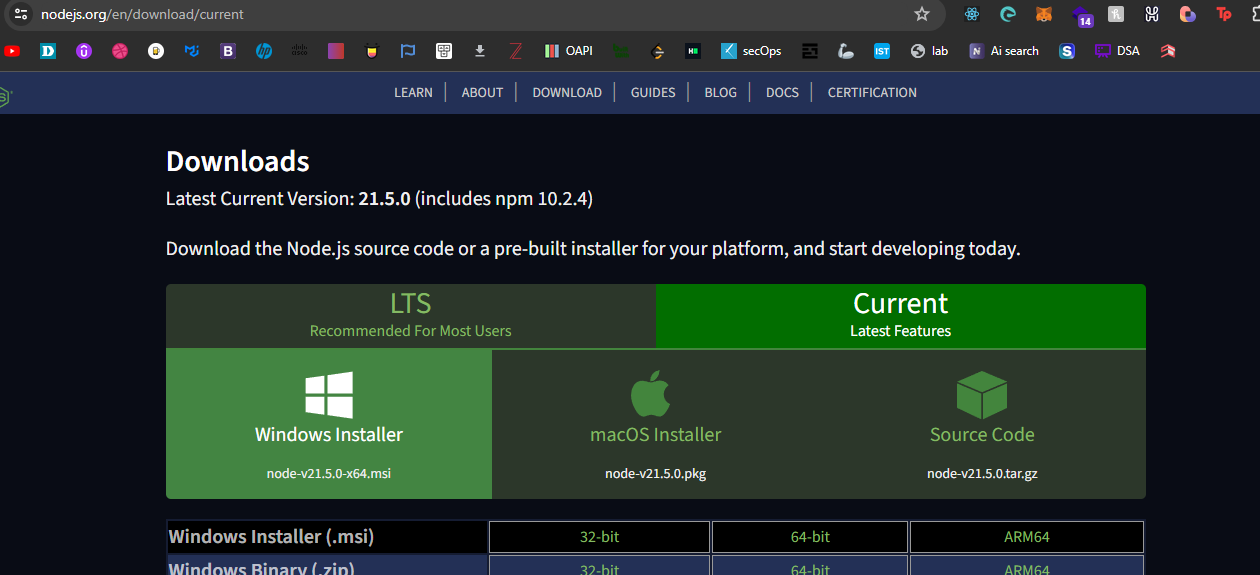
1. **Define cookies.**
2. **What is session mechanism?**
3. **How to delete cookies.**
4. **Define Http session.**
5. **How to set cookies.**

**Experiment 9:**

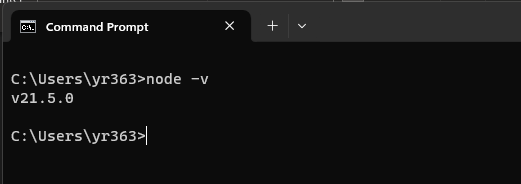
**Create a custom server using http module and explore the other modules of Node JS like OS.path ,event**

**NodeJs Installation Steps**

1. Download the Node.js installer from the official website.



1. Run the downloaded .msi file2.
2. Follow the setup wizard instructions, accept the End-User License Agreement (EULA), set the destination folder, and choose the components to install.
3. Complete the installation.
4. Verify the installation by opening your command prompt or Windows Powershell and running the command node -v



Now Installation of node is done.

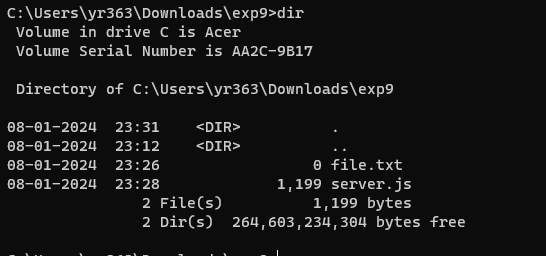
To create custom server we need to write a javascript code, mostly we use file name server.js or index.js

**Server.js**

**const** http = require('http');  
**const** os = require('os');  
**const** path = require('path');  
**const** EventEmitter = require('events');  
  
**const** server = http.createServer((req, res) => {  
 res.writeHead(200, {'Content-Type': 'text/plain'});  
 res.end('Hello, this is your custom server!\n');  
});  
  
**const** PORT = 3000;  
server.listen(PORT, () => {  
 console.log(`Server running at http://localhost:${PORT}/`);  
});  
  
// os:  
console.log('OS Platform:', os.platform());  
console.log('CPU Architecture:', os.arch());  
console.log('Free Memory:', os.freemem());  
console.log('Total Memory:', os.totalmem());  
  
// path   
**const** filePath = 'C:/Users/yr363/Downloads/exp9/file.txt'; //replace the path with your system path  
console.log('File Name:', path.basename(filePath));  
console.log('Directory Name:', path.dirname(filePath));  
console.log('File Extension:', path.extname(filePath));  
  
// events:  
**class** MyEmitter **extends** EventEmitter {}  
  
**const** myEmitter = **new** MyEmitter();  
  
myEmitter.on('customEvent', (arg) => {  
 console.log('Custom Event Occurred:', arg);  
});  
  
myEmitter.emit('customEvent', 'This is an argument for the event');

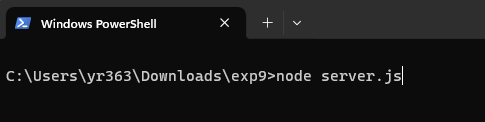
Now We have created our server.js file to run the server using node we need to follow:

1. Open a terminal or command prompt in the directory where the server.js file is located. (In My Case it is *“C:\Users\yr363\Downloads\exp9”*).



1. Run the following command to start the server:

*node server.js*

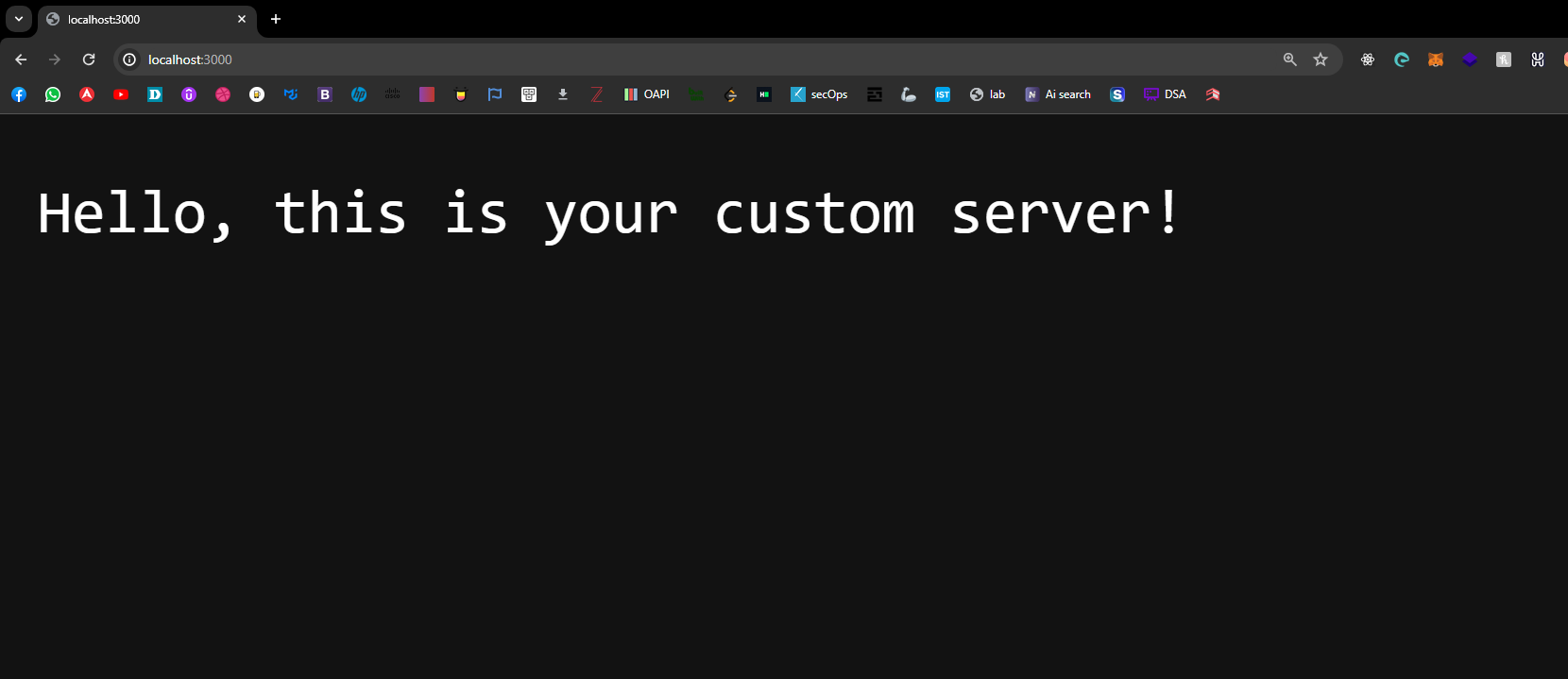


OUTPUT:

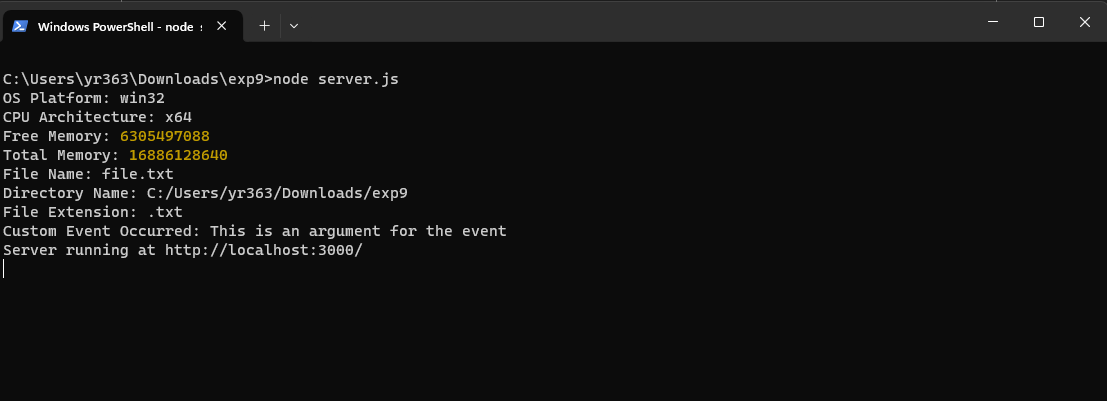


Now our server is running

Below is the response came from the server after it is running without any error



Below it shows that the server is running in localhost:3000 as we used os, path , events as we run the server we will get the stats of our system , path of a file we want to know and event occurred



The last line “Sever running at <http://localhost:3000/>” refers that our server is running on the port number 3000

**Viva Questions:**

1. What is Node.Js ?Where can we use it.
2. How dose Node.js work?
3. Write the steps to install Node.js.
4. Difference between angular and node.js
5. Define server.

**Experiment 10.**

**Develop an express web application that can interact with REST API to perform CRUD operation on student data(Use Postman)**

1)create project

2) create application

3) create urls.py application level.- include that in project level urls

4) go through django-restframework - install it in terminal

- include in settings in apps- 'rest\_framework'

5) create urls.py at app level and link with views

6)start writing models in the app level to link with database

# this is for linking with api

class Student(models.Model):

name = models.CharField(max\_length=100, null=False)

email = models.CharField(max\_length=100, null=False)

phone = models.Charfield(max\_length=100, null=False)

def \_str\_(self):

return self.name

7) python manage.py migrate, python manage.py makemigrations

8)add that models in admin.py at application level

from .models import Student.

# Register your models here.

admin.site.register(Student)

9)python manage.py createsuperuser

username:

password:

10)check in the admin database

and add students data

check with models and admin and apply python manage.py makemigrations and python manage.py migrate

11) write views, link models in the views and write views

12) create a serializers.py to to convert complex data types, such as Django model instances, into Python data types that can be easily rendered into JSON, XML, or other content types.

13) write serializers.py and include that in views.py

14) import modules of api\_view from django\_restframework

- now calling apiviews in decorators format

15) first api view of get method in views is for get api information of student data from adminofdjango

16) write api views for get method in views for get apiinfor of student individual data using id from admin, create a urls.py also for that

17)add a new student in views.py from serializers and ,create a urls.py for calling

18) for updating the data ,create a post api view and ,create a urls.py for calling

19) for deleting the data , create a delete api view ,no need of serializers calling here , and call that in urls.py

20) go through post man for api check

postman

create a workspace

raw and add json to update, delete,add students

content type, application/json and raw to view json in views.

**This is views.py:**

from rest\_framework.response import Response

from rest\_framework.decorators import api\_view

from api.models import Student

# Create your views here.

from .serializers import StudentSerializer

@api\_view(['GET'])

def index(request):

students = Student.objects.all()

serialstudents = StudentSerializer(students, many=True)

#Return a Response

return Response(serialstudents.data)

# Or If you want to specify custom status code along with the data, then follow the below structure

return Response({

'status':200,

'students':serialstudents.data

})

# fetch the student data individually ,we will use again get method

@api\_view(['GET'])

def studentView(request, pk):

try :

student = Student.objects.get(id=pk)

serialstudent = StudentSerializer(student, many=False)

return Response({

'status':200,

'students':serialstudent.data,

})

except :

return Response({'status':400})

@api\_view(['POST'])

def studentAdd(request):

try:

serialdata = StudentSerializer(data=request.data)

if serialdata.is\_valid():

serialdata.save()

return Response({

'status':200,

'student':serialdata.data,

'message':'Student added successfully'

})

except:

return Response({'status':400})

@api\_view(['POST'])

def studentUpdate(request, pk):

try :

student = Student.objects.get(id=pk)

serialstudent = StudentSerializer(instance=student, data=request.data)

if serialstudent.is\_valid():

serialstudent.save()

return Response({

'status':200,

'student':serialstudent.data,

'message':'Updated successfully'

})

except :

return Response({'status':400})

@api\_view(['DELETE'])

def studentdelete(request, pk):

try:

student = Student.objects.get(id=pk)

student.delete()

students = Student.objects.all()

serialstudents = StudentSerializer(students, many=True)

return Response({

'status':200,

'student':serialstudents.data,

'message':'Student Deleted successfully'

})

except:

return Response({'status':400})

**This is models.py :**

from django.db import models

# Create your models here.

class Student(models.Model):

name = models.CharField(max\_length=191, null=True)

age = models.CharField(max\_length=20, null=True)

course = models.CharField(max\_length=191, null=True)

phone = models.CharField(max\_length=191, null=True)

email = models.CharField(max\_length=191, null=True)

dateCreated = models.DateTimeField(auto\_now\_add=True, null=True)

def \_str\_(self): #this is to show the name in the admin panel, you will understand in the next tutorial

return self.name

**create an serializers.py file at application level**

**This is serializer.py:**

from rest\_framework import serializers

from .models import Student

class StudentSerializer(serializers.ModelSerializer):

class Meta:

model = Student

fields = '\_all\_'

**This is admin.py:**

from django.contrib import admin

from .models import Student

# Register your models here.

admin.site.register(Student)

Then,

Python manage.py migrate

Python manage.py makemigrations

Python manage.py createsuperuser

This is urls.py in app :

from django.urls import path

from . import views

urlpatterns = [

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

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

path('student-view/<str:pk>/', views.studentView, name="studentview"),

path('add-student/', views.studentAdd, name="studentadd"),

path('update-student/<str:pk>/', views.studentUpdate, name="studentupdate"),

path('delete-student/<str:pk>/', views.studentdelete, name="studentdelete")

]

**This is urls.py in project:**

from django.contrib import admin

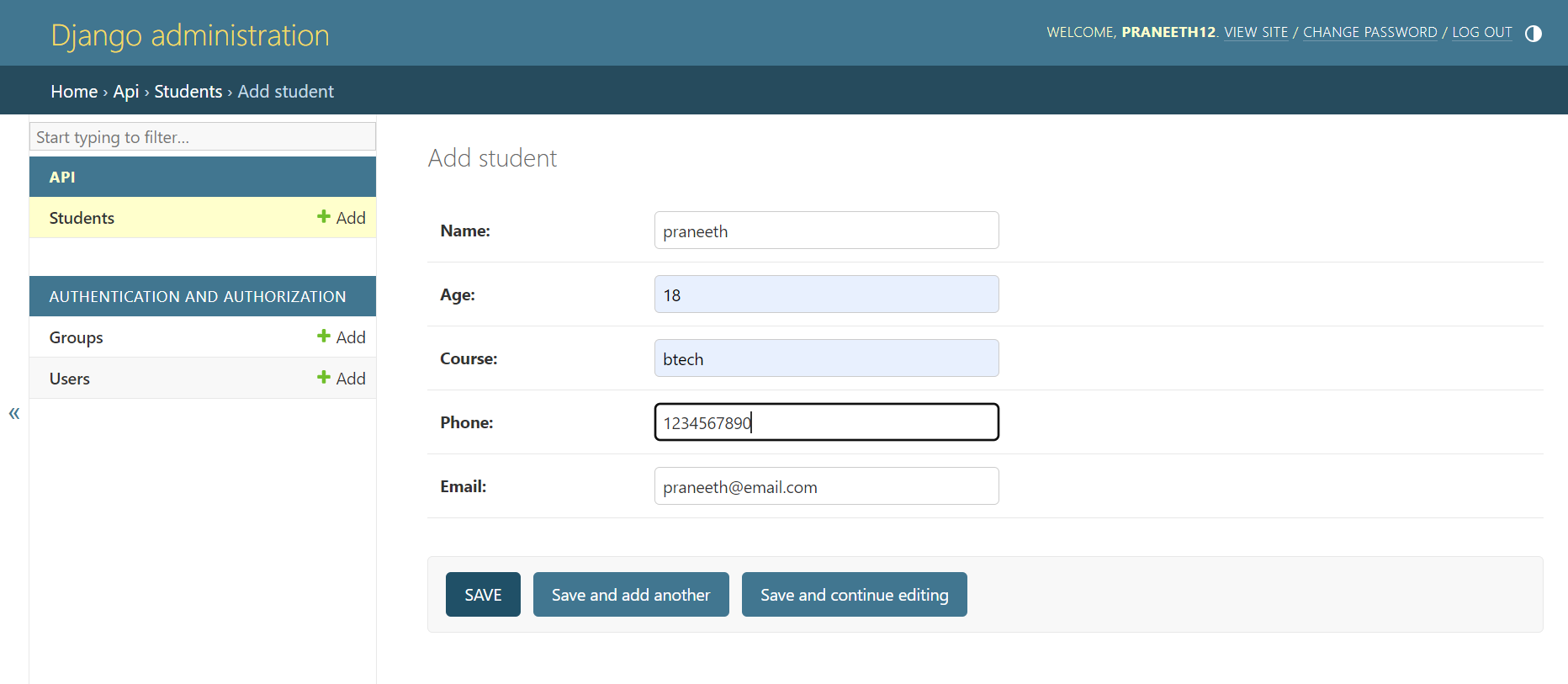
from django.urls import path, include

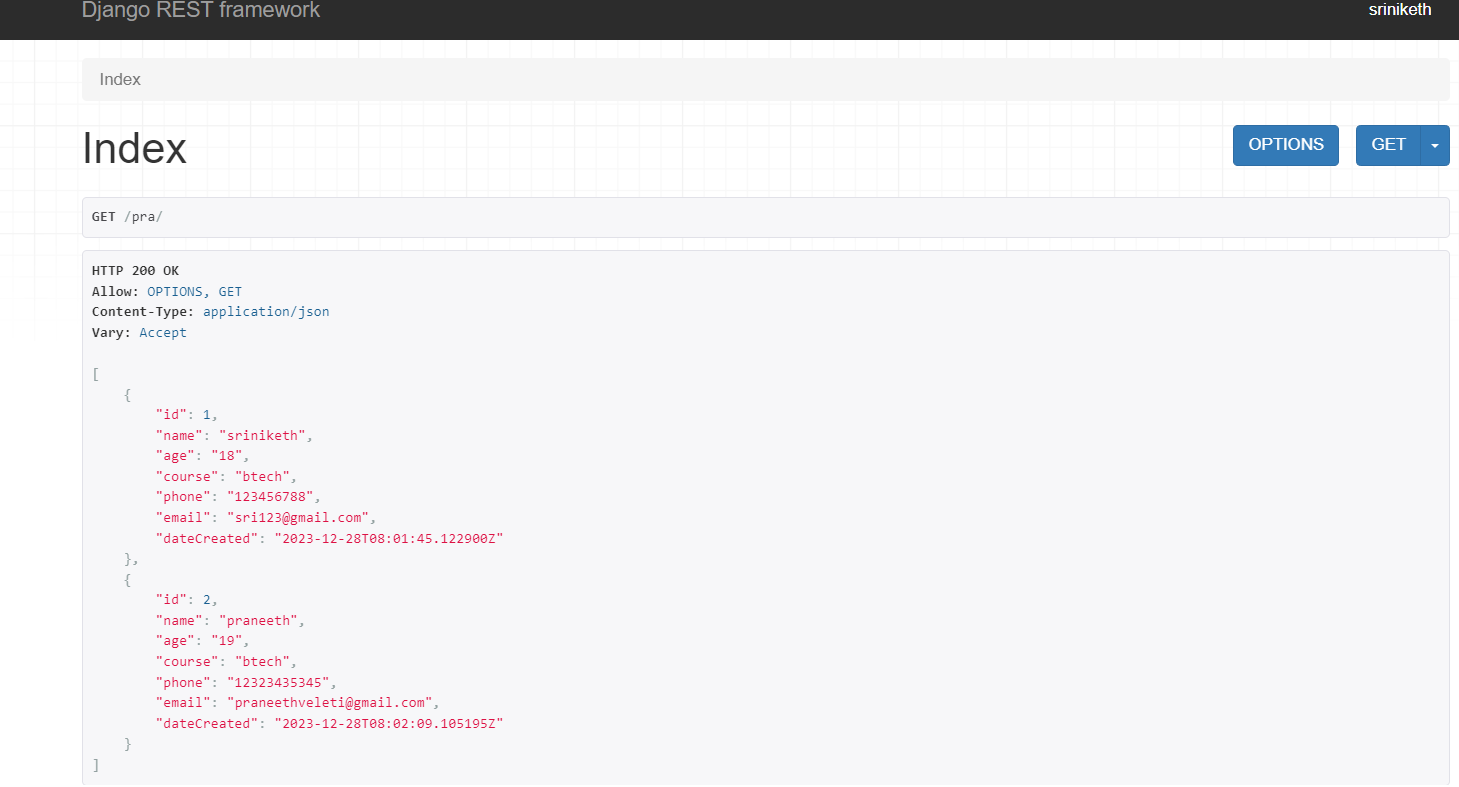
urlpatterns = [

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

path('/', include('api.urls')),

]





**Viva Questions:**

1)What is express web application.

2) What Rest Api.

3)Write short note on postman api.

4) Explain the below statement

urlpatterns = [

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

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

path('student-view/<str:pk>/', views.studentView, name="studentview"),

path('add-student/', views.studentAdd, name="studentadd"),

path('update-student/<str:pk>/', views.studentUpdate, name="studentupdate"),

path('delete-student/<str:pk>/', views.studentdelete, name="studentdelete")

]

5) How to create a serializers.py to convert complex data types.

**Experiment 11:**

**For the above application create authorized end points using JWT(JSON Web Token).**

1) create a project

2) create an application

3) in app/models.py create email and password fields in place of admin login .

add auth\_user\_model in settings.py

4)include that in installed apps.

5)python manage.py makemigrations,

python manage.py migrate

python manage.py makemigrations appname

6)make changes in admin.py at application level

7)create superuser

8) once check with the server .and login into the admin and go through users .

9)Next Step is to install Django Rest Framework in your Project Directory. Django Rest Framework (DRF) is a toolkit build in web application which is used for creating web API’s. It returns web API’s in the form of raw JSON. Install DRF using following command.

-->pip install djangorestframework

-->add rest\_framework in settings.py

10) create serializers.py in app

11) define the views in application

12)define urls for the views in application

13) add urls of application at project level

14)now we will use simplejwt to login user

We will use simple JWT to login user an generate access and refresh Token for authentication of user. Install simple JWT in your project directory with the pip command.

-->pip install djangorestframework-simplejwt

15) add rest\_framework\_simplejwt in installed apps of settings

16)configure jwt at settings

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

),

}

16) lastly, In our app level urls.py we’ll add routes for simple jwt as TokenObtainPairView and TokenRefreshView views.

17) use postman

first register using post request- because to store into database

-body

-form-data

18) after registrering login we will get

two tokens access and refresh token

19)

Access Token : Access token is the encoded string which contains information about user, permissions etc. Token are used as a bearer token, Bearer means which hold data in it. An access token is put in the Authorization header of your request for the user’s API.

Refresh Token : An access tokens have very short life span because of the security purpose. When it expires a user need to generated new access token for authentication. So refresh token is used to request new access tokens without user interaction.

20)

We can also customize the behavior of simple JWT by changing some of the settings variables in the settings.py. Copy and paste the following code in settings.py file.

**THIS IS VIEWS.PY:**

from django.shortcuts import render

from rest\_framework.views import APIView

from .serializers import UserSerializer

from rest\_framework.response import Response

# view for registering users

class RegisterView(APIView):

    def post(self, request):

        serializer = UserSerializer(data=request.data)

        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response(serializer.data)

THIS IS URLS.PY IN APP LEVEL:

from django.urls import path

from .views import RegisterView

from rest\_framework\_simplejwt.views import (

    TokenObtainPairView,

    TokenRefreshView,

)

urlpatterns = [

    path('api/login/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

    path('api/login/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

    path('api/register/', RegisterView.as\_view(), name="sign\_up"),

]

THIS IS URLS.PY IN PROJECT LEVEL:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('account/', include('myapp.urls')),

]

THIS IS MODEL.PY:

from django.db import models

# Create your models here.

#from django.db import models

from django.contrib.auth.models import AbstractUser, BaseUserManager

class UserManager(BaseUserManager):

    use\_in\_migration = True

    def create\_user(self, email, password=None, \*\*extra\_fields):

        if not email:

            raise ValueError('Email is Required')

        user = self.model(email=self.normalize\_email(email), \*\*extra\_fields)

        user.set\_password(password)

        user.save(using=self.\_db)

        return user

    def create\_superuser(self, email, password, \*\*extra\_fields):

        extra\_fields.setdefault('is\_staff', True)

        extra\_fields.setdefault('is\_superuser', True)

        extra\_fields.setdefault('is\_active', True)

        if extra\_fields.get('is\_staff') is not True:

            raise ValueError('Superuser must have is\_staff = True')

        if extra\_fields.get('is\_superuser') is not True:

            raise ValueError('Superuser must have is\_superuser = True')

        return self.create\_user(email, password, \*\*extra\_fields)

class UserData(AbstractUser):

    username = None

    name = models.CharField(max\_length=100, unique=True)

    email = models.EmailField(max\_length=100, unique=True)

    date\_joined = models.DateTimeField(auto\_now\_add=True)

    is\_admin = models.BooleanField(default=False)

    is\_active = models.BooleanField(default=True)

    is\_staff = models.BooleanField(default=False)

    is\_superuser = models.BooleanField(default=False)

    objects = UserManager()

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = ['name']

    def \_str\_(self):

        return self.name

THIS IS ADMIN.PY:

from django.contrib import admin

from .models import UserData

admin.site.register(UserData)

THIS IS SERIALIZERS.PY:

from rest\_framework import serializers

from .models import UserData

class UserSerializer(serializers.ModelSerializer):

    class Meta:

        model = UserData

        fields = ["id", "email", "name", "password"]

    def create(self, validated\_data):

        user = UserData.objects.create(email=validated\_data['email'],

                                       name=validated\_data['name']

                                         )

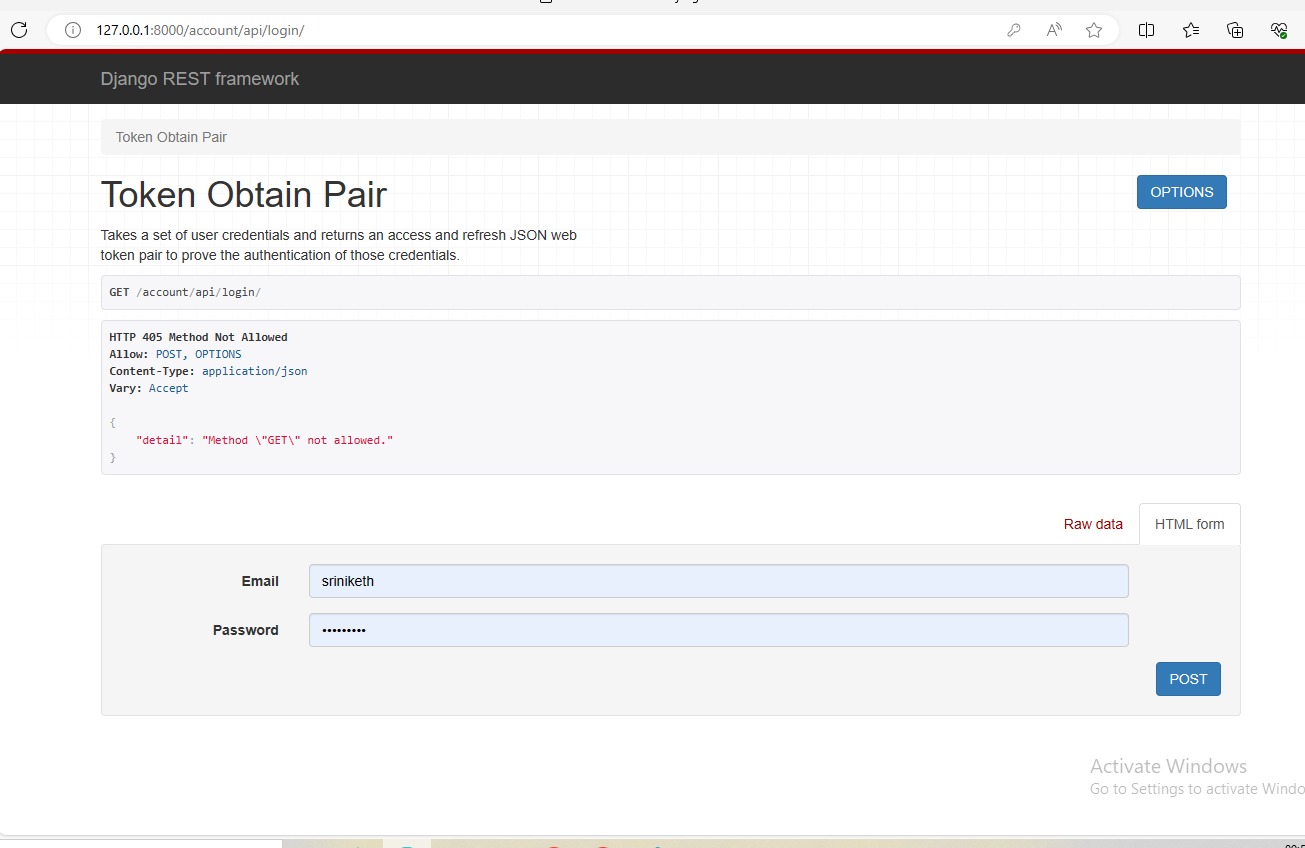
        user.set\_password(validated\_data['password'])

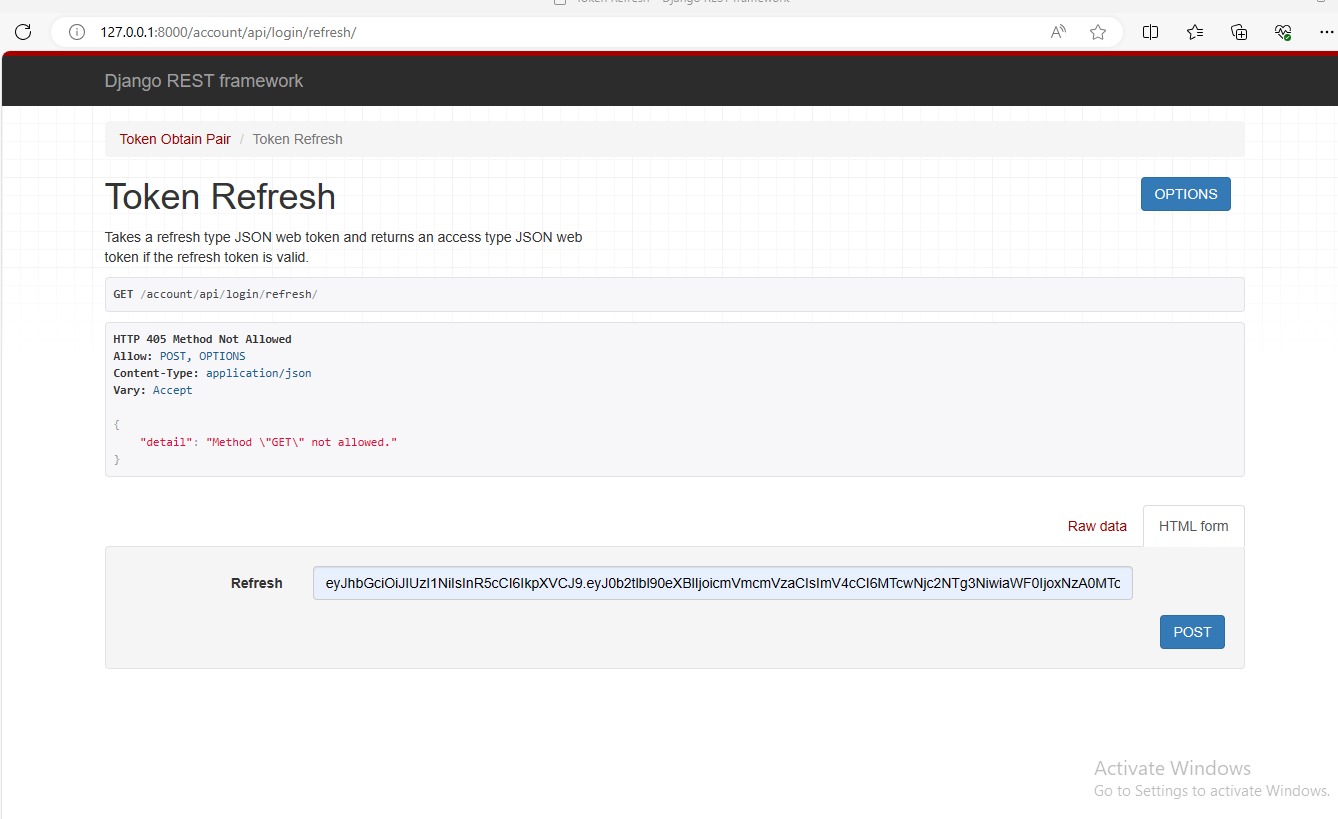
        user.save()

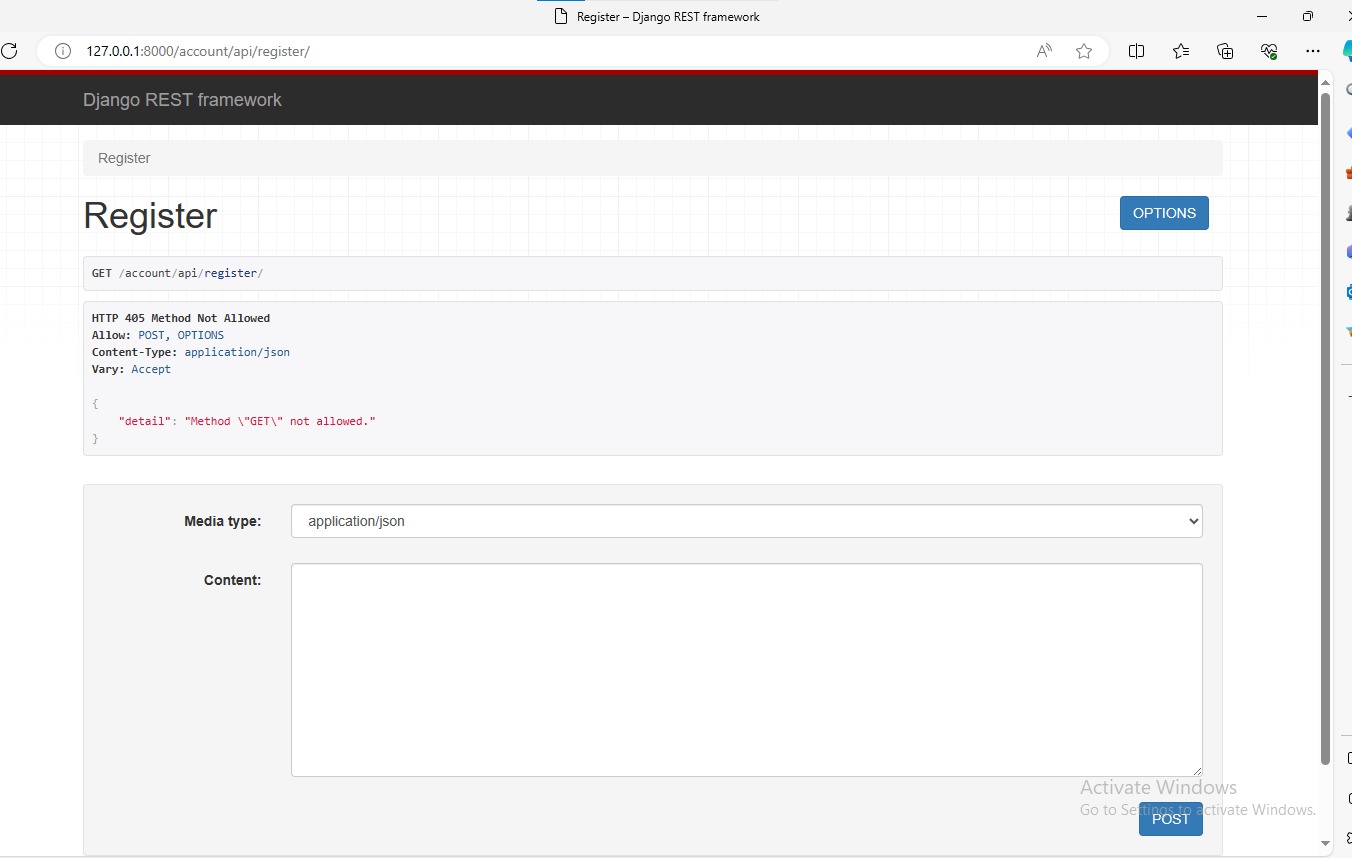
        return user

In seetings.py include the app name in installed apps and also add the rest\_framework in it

Outputs:







**Viva Questions**

1. What is JWT.
2. What is the need for JWT.
3. What are advantages of JWT.
4. What is structure of JWT.
5. How to install Django Rest Framework in your Project Directory

**Experiment 12:**

**Create a react application for the student management system having registration, login, contact, about pages and implement routing to navigate through these pages.**

Creating a React application for a student management system involves several steps. Below is a basic example to get you started with registration, login, contact, and about pages using React Router for navigation.

Firstly, make sure you have Node.js installed on your machine. Then, follow these steps:

1. \*\*Create React App:\*\*

Open your terminal and run the following command to create a new React app:

**npx create-react-app student-management-system**

2. \*\*Install React Router:\*\*

Navigate to your project folder:

**cd student-management-system**

Install React Router:

**npm install react-router-dom**

3. \*\*Create Components:\*\*

Inside the `src` folder, create components for `Registration`, `Login`, `Contact`, and `About`. For instance, in the `src` folder, create files like `Registration.js`, `Login.js`, `Contact.js`, and `About.js`. These files will contain your component logic and JSX.

4. \*\*Set Up Routing:\*\*

Create a file called `AppRouter.js` to handle routing:

**import React from 'react';**

**import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';**

**import Registration from './Registration';**

**import Login from './Login';**

**import Contact from './Contact';**

**import About from './About';**

**const AppRouter = () => {**

**return (**

**<Router>**

**<Switch>**

**<Route exact path="/" component={Login} />**

**<Route path="/registration" component={Registration} />**

**<Route path="/contact" component={Contact} />**

**<Route path="/about" component={About} />**

**</Switch>**

**</Router>**

**);**

**};**

**export default AppRouter;**

1. \*\*Implement Navigation:\*\*

Update your `App.js` to use the `AppRouter` component:

**import React from 'react';**

**import AppRouter from './AppRouter';**

**function App() {**

**return (**

**<div className="App">**

**<AppRouter />**

**</div>**

**);**

**}**

**export default App;**

6. \*\*Add Navigation Links:\*\*

In each component (`Login.js`, `Registration.js`, etc.), use `Link` from `react-router-dom` to navigate between pages:

**import React from 'react';**

**import { Link } from 'react-router-dom';**

**const Login = () => {**

**return (**

**<div>**

**<h1>Login Page</h1>**

**{/\* Add Link to other pages \*/}**

**<Link to="/registration">Register</Link>**

**<Link to="/contact">Contact</Link>**

**<Link to="/about">About</Link>**

**</div>**

**);**

**};**

**export default Login;**

7. \*\*Run the App:\*\*

Start your React app:

**npm start**

This will launch your application in the browser. You should be able to navigate between the pages using the links provided.

**Viva Questions:**

1. What is React and JSX?
2. What are the major features of React?
3. How to create components in React.
4. What are the advantages of using React?
5. What are the limitations of React?

**Output:**

**Experiment 13:**

**Create a service in react that fetches the weather information from open weathermap.org and the display the current and historical weather information using graphical representation using chart.js**

Certainly! To create a service that fetches weather information from OpenWeatherMap and displays it using Chart.js in a React app, follow these steps:

1. \*\*Setup a React App:\*\*

Create a new React app or use an existing one.

2. \*\*Install dependencies:\*\*

Install `axios` for making API requests and `chart.js` for chart visualization:

**npm install axios chart.js**

3. \*\*Create a WeatherService:\*\*

Create a file named `WeatherService.js` to handle API requests:

**import axios from 'axios';**

**const API\_KEY = 'YOUR\_OPENWEATHERMAP\_API\_KEY';// here keep the api key from openweathermap**

**const WeatherService = {**

**getCurrentWeather: async (city) => {**

**try {**

**const response = await axios.get(**

**`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API\_KEY}`**

**);**

**return response.data;**

**} catch (error) {**

**throw new Error('Error fetching current weather data');**

**}**

**},**

**getHistoricalWeather: async (city) => {**

**try {**

**const response = await axios.get(**

**`https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${API\_KEY}`**

**);**

**return response.data;**

**} catch (error) {**

**throw new Error('Error fetching historical weather data');**

**}**

**},**

**};**

**export default WeatherService;**

4. \*\*Create a Weather component:\*\*

Create a `Weather.js` file to display weather information using Chart.js:

**import React, { useState, useEffect } from 'react';**

**import Chart from 'chart.js/auto';**

**import WeatherService from './WeatherService';**

**const Weather = ({ city }) => {**

**const [currentWeather, setCurrentWeather] = useState(null);**

**const [historicalWeather, setHistoricalWeather] = useState(null);**

**useEffect(() => {**

**WeatherService.getCurrentWeather(city)**

**.then((data) => setCurrentWeather(data))**

**.catch((error) => console.error(error));**

**WeatherService.getHistoricalWeather(city)**

**.then((data) => setHistoricalWeather(data))**

**.catch((error) => console.error(error));**

**}, [city]);**

**useEffect(() => {**

**if (historicalWeather) {**

**// Process historical weather data and create chart**

**createChart();**

**}**

**}, [historicalWeather]);**

**const createChart = () => {**

**// Extract necessary data from historicalWeather and create a Chart using Chart.js**

**// Example: Create a line chart for temperature over time**

**const ctx = document.getElementById('weatherChart');**

**if (ctx) {**

**new Chart(ctx, {**

**type: 'line',**

**data: {**

**labels: historicalWeather.list.map((item) => item.dt\_txt),**

**datasets: [**

**{**

**label: 'Temperature (°C)',**

**data: historicalWeather.list.map((item) => item.main.temp),**

**fill: false,**

**borderColor: 'rgb(75, 192, 192)',**

**tension: 0.1,**

**},**

**],**

**},**

**});**

**}**

**};**

**return (**

**<div>**

**<h2>Current Weather</h2>**

**{/\* Display current weather information \*/}**

**{/\* You can format the data as needed \*/}**

**{currentWeather && (**

**<div>**

**<p>Temperature: {currentWeather.main.temp} °C</p>**

**<p>Description: {currentWeather.weather[0].description}</p>**

**{/\* Add other necessary current weather information \*/}**

**</div>**

**)}**

**<h2>Historical Weather Chart</h2>**

**{/\* Display chart for historical weather \*/}**

**<canvas id="weatherChart" width="400" height="200"></canvas>**

**</div>**

**);**

**};**

**export default Weather;**

5. \*\*Use the Weather component:\*\*

In your main App.js or any other relevant component, import and use the `Weather` component:

**import React from 'react';**

**import Weather from './Weather';**

**const App = () => {**

**return (**

**<div className="App">**

**<Weather city="YOUR\_CITY\_NAME" /> {/\* Replace with desired city \*/}**

**</div>**

**);**

**};**

**export default App;**

Make sure to replace `'YOUR\_OPENWEATHERMAP\_API\_KEY'` with your actual API key from OpenWeatherMap, and `'YOUR\_CITY\_NAME'` with the desired city for weather information retrieval.

**OUTPUT**

**Viva Questions**

1. What is use state Hook?
2. What is use effect Hook?
3. How to create weather component.
4. What is axios?
5. What are the rules that must be followed while using React Hooks?

**Experiment 14:**

**Create a TODO application in react with necessary components and deploy it into github.**

todo list project:

from django.shortcuts import render, redirect

from django.contrib import messages

# import todo form and models

from .forms import TodoForm

from .models import Todo

def index(request):

item\_list = Todo.objects.order\_by("-date")

if request.method == "POST":

form = TodoForm(request.POST)

if form.is\_valid():

form.save()

return redirect('todo')

form = TodoForm()

page = {

"forms": form,

"list": item\_list,

"title": "TODO LIST",

}

return render(request, 'todoapp/index.html', page)

def remove(request, item\_id):

item = Todo.objects.get(id=item\_id)

item.delete()

messages.info(request, "item removed !!!")

return redirect('todo')

**models.py:**

from django.db import models

from django.utils import timezone

class Todo(models.Model):

title = models.CharField(max\_length=100)

details = models.TextField()

date = models.DateTimeField(default=timezone.now)

def \_str\_(self):

return self.title

no need to create application level urls.py ,directly do code in project level urls.py

from django.contrib import admin

from django.urls import path

from todoapp import views

urlpatterns = [

path('', views.index, name="todo"),

# pass item\_id as primary key to remove that the todo with given id

path('del/<str:item\_id>', views.remove, name="del"),

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

]

Create the forms.py in app level :

from django import forms

from .models import Todo

class TodoForm(forms.ModelForm):

class Meta:

model = Todo

fields = "\_all\_"

create templates folder in application and create subfolder appname, in that create a file index.html

Index.html:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>{{title}}</title>

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

<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>

<!--style-->

<style>

.card {

box-shadow: 0 4px 8px 0 rgba(0,0,0,0.5),

0 6px 20px 0 rgba(0,0,0,0.39);

background: orange;

margin-bottom : 5%;

border-radius: 25px;

padding : 2%;

overflow: auto;

resize: both;

text-overflow: ellipsis;

}

.card:hover{

background: red;

}

.submit\_form{

text-align: center;

padding: 3%;

background: pink;

border-radius: 25px;

box-shadow: 0 4px 8px 0 rgba(0,0,0,0.4),

0 6px 20px 0 rgba(0,0,0,0.36);

}

</style>

</head>

<body class="container-fluid">

{% if messages %}

{% for message in messages %}

<div class="alert alert-info">

<strong>{{message}}</strong>

</div>

{% endfor %}

{% endif %}

<center class="row">

<h1><i>TODO LIST</i></h1>

<hr />

</center>

<div class="row">

<div class="col-md-8">

{% for i in list %}

<div class="card">

<center><b>{{i.title}}</b></center>

<hr/>

{{i.date}}

<hr/>

{{i.details}}

<br />

<br />

<form action="/del/{{i.id}}" method="POST" style=" padding-right: 4%; padding-bottom: 3%;">

{% csrf\_token %}

<button value="remove" type="submit" class="btnbtn-primary" style="float: right;"><span class="glyphiconglyphicon-trash"></span> remove</button>

</form>

</div>

{% endfor%}

</div>

<div class="col-md-1"></div>

<div class="col-md-3" >

<div class="submit\_form">

<form method="POST">

{% csrf\_token %}

{{forms}}

<center>

<input type="submit" class="btnbtn-default" value="submit" />

</center>

</form>

</div>

</div>

</div>

</body>

</html>

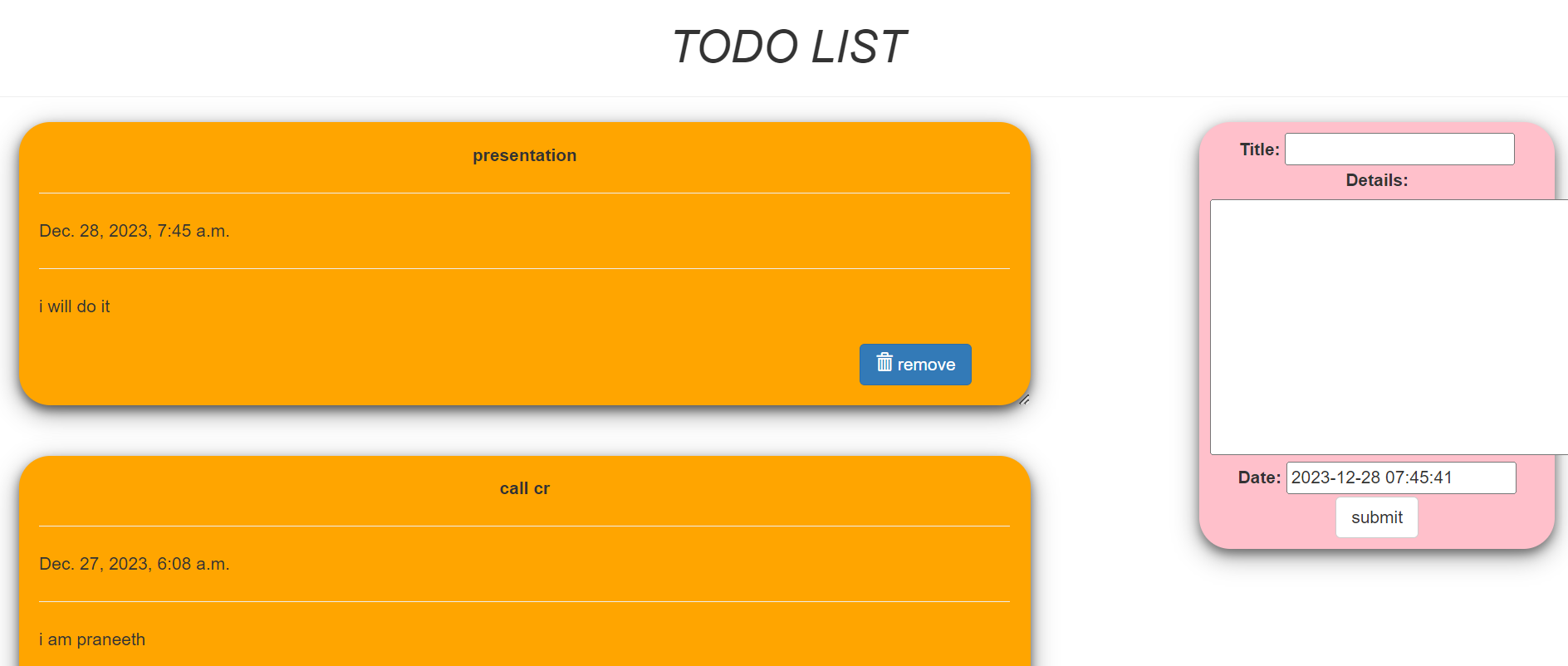
admin.py:

from django.contrib import admin

from .models import Todo

# Register your models here.

admin.site.register(Todo)



**Viva Questions:**

1.How to create TODO list.

2.Explain the below code

def index(request):

item\_list = Todo.objects.order\_by("-date")

if request.method == "POST":

form = TodoForm(request.POST)

if form.is\_valid():

form.save()

return redirect('todo')

form = TodoForm()

page = {

"forms": form,

"list": item\_list,

"title": "TODO LIST",

}

return render(request, 'todoapp/index.html', page)

3.what is <form method="POST">.

4. Explain

<form action="/del/{{i.id}}" method="POST" style=" padding-right: 4%; padding-bottom: 3%;">

5.What is react.