

SUBJECT: FULLSTACK DEVELOPMENT (21CS62)

LAB COMPONENT SOLUTIONS

Module-1: MVC based Web Designing

Laboratory Component:

1. Installation of Python, Django and Visual Studio code editors can be demonstrated.

Python download and installation Link:

<https://www.python.org/downloads/>

Visual Studio Code download and installation link:

<https://code.visualstudio.com/>

Django installation:

Open a command prompt and type following command:

```
pip install django
```

2. Creation of virtual environment, Django project and App should be demonstrated

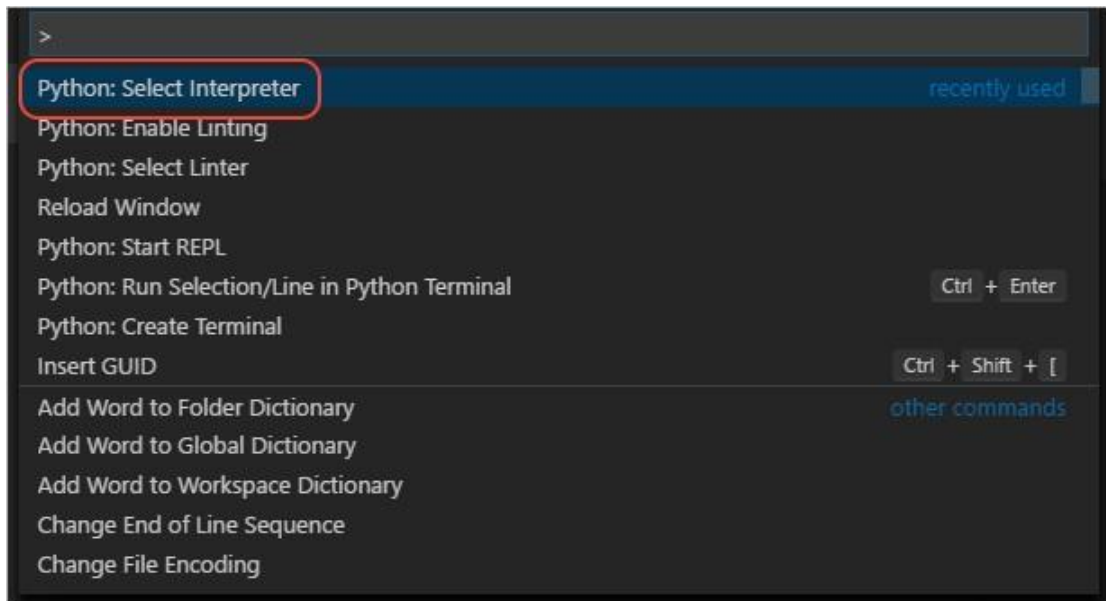
Follow these steps

1. Install the [Python extension](#). - Open VS Code IDE and click extensions there automatically u will be shown Python extension (Make sure you are connected to Internet)
2. On your file system, create a project folder
3. In that folder, use the following command (as appropriate to your computer) to create a virtual environment named `env` based on your current interpreter:

```
# Windows
```

```
python -m venv env
```

4. Open the project folder in VS Code by running `code .`, or by running VS Code and using the **File > Open Folder** command.
5. In VS Code, open the Command Palette (**View > Command Palette** or (**Ctrl+Shift+P**)). Then select the **Python: Select Interpreter** command:



6. The command presents a list of available interpreters that VS Code can locate automatically (your list will vary; if you don't see the desired interpreter, see [Configuring Python environments](#)). From the list, select the virtual environment in your project folder that starts with `./env` or `.\env`:
7. Create a New Terminal : In Menu Terminal -> New Terminal option

Creating project:

1. Create a django project -

Type following command in the terminal opened:

`django-admin startproject p .`

(dot following project name is important which refers to current directory)

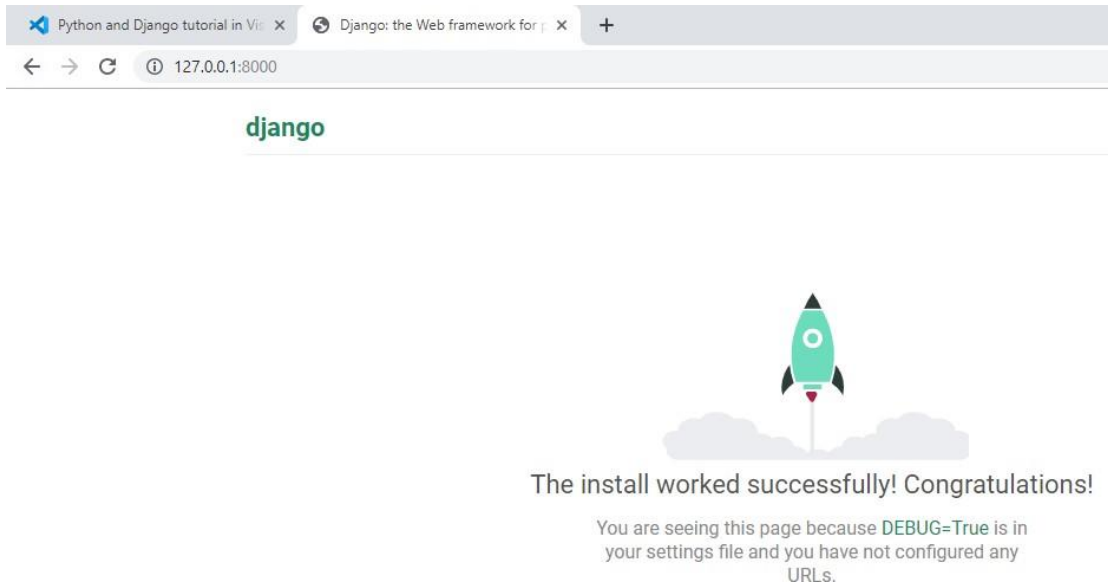
This `startproject` command assumes (by use of `.` at the end) that the current folder is your project folder, and creates the following within it:

- `manage.py`: The Django command-line administrative utility for the project. You run administrative commands for the project using `python manage.py <command> [options]`.
- A subfolder named `p` which contains the following files:
 - `__init__.py`: an empty file that tells Python that this folder is a Python package.
 - `wsgi.py`: an entry point for WSGI-compatible web servers to serve your project. You typically leave this file as-is as it provides the hooks for production web servers.
 - `settings.py`: contains settings for Django project, which you modify in the course of developing a web app.
 - `urls.py`: contains a table of contents for the Django project, which you also modify in the course of development.
- 2. To verify the Django project, make sure your virtual environment is activated, then start Django's development server using the

command `python manage.py runserver`. The server runs on the default port 8000, and you see output like the following output in the terminal window:

Verify server by typing:

```
python manage.py runserver
```



When you run the server the first time, it creates a default SQLite database in the file `db.sqlite3`, which is intended for development purposes but can be used in production for low-volume web apps. Also, Django's built-in web server is intended *only* for local development purposes. When you deploy to a web host, however, Django uses the host's web server instead. The `wsgi.py` module in the Django project takes care of hooking into the production servers.

If you want to use a different port than the default 8000, specify the port number on the command line, such as `python manage.py runserver 5000`.

3. When you're done, close the browser window and stop the server in VS Code using `Ctrl+C` as indicated in the terminal output window.
4. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your project folder (where `manage.py` resides):

```
python manage.py startapp lab1
```

5. The command creates a folder called `lab1` that contains a number of code files and one subfolder. Of these, you frequently work with `views.py` (that contains the functions that define pages in your web app) and `models.py` (that contains classes defining your data objects). The `migrations` folder is used by Django's administrative utility to manage database versions. There are also the files `apps.py` (app configuration), `admin.py` (for creating an administrative interface), and `tests.py` (for unit tests).

3. Develop a Django app that displays current date and time in server

In lab1 subfolder, make following changes to views.py:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

# Create your views here.
import datetime
def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body><h1>It is now %s.</h1></body></html>" % now
    return HttpResponseRedirect(html)
```

In project named first, make following changes to urls.py

```
from django.contrib import admin
from django.urls import path
from lab11.views import current_datetime
urlpatterns = [
```

```
    path('cdt/', current_datetime),
```

```
]
```

Output:



It is now 2024-02-03 18:42:01.631243.

4. Develop a Django app that displays date and time four hours ahead and four hours before as an offset of current date and time in server.

In lab11 subfolder, make following changes to views.py:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
```

```
# Create your views here.
import datetime
def current_datetime(request):
    now = datetime.datetime.now()
```

```

html = "<html><body><h1>It is now %s.</h1></body></html>" % now
return HttpResponse(html)
def four_hours_ahead(request):

    dt = datetime.datetime.now() + datetime.timedelta(hours=4)
    html = "<html><body><h1>After 4hour(s), it will be %s.</h1>"% (dt,)
    return HttpResponse(html)

def four_hours_before(request):

    dt = datetime.datetime.now() + datetime.timedelta(hours=-4)
    html = "<html><body><h1>Before 4 hour(s), it was %s.</h1>"% (dt,)
    return HttpResponse(html)

```

In project named first, make following changes to urls.py

```

from django.contrib import admin
from django.urls import path
from lab11.views import current_datetime, four_hours_ahead, four_hours_before
urlpatterns = [

    path('cdt/', current_datetime),
    path('fhrsa/', four_hours_ahead),
    path('fhrsb/', four_hours_before),

]

```

Output:

← → ↻ ⓘ 127.0.0.1:8000/fhrsa/

After 4hour(s), it will be 2024-02-03 22:43:50.544397.

← → ↻ ⓘ 127.0.0.1:8000/fhrrsb/

Before 4 hour(s), it was 2024-02-03 14:44:10.994024.

LAB COMPONENT SOLUTIONS

Module-1: Additional Programs on Django Views and URLs *Develop a Django app that displays tables of squares of pairs of numbers input in the URL.*

Views.py

```
from datetime import date
from django.http import HttpResponse
from django.shortcuts import render
from django.template import Context, Template
def create_table_of_squares(request,s,n):
    result=""
    for i in range(1,n+1):
        result+="

" +str(s)+"*" +str(i)+"="+str((s*i))+"</p>"
    return HttpResponse(result)


```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import create_table_of_squares
urlpatterns = [
    path('admin/', admin.site.urls), path('cts/<int:s>/<int:n>',
    create_table_of_squares),

]
```

← → ↺ 127.0.0.1:8000/cts/3/6

Table of squares

3*1=3

3*2=6

3*3=9

3*4=12

Output

Develop a Django app that displays number of vowels and consonants and also list of vowels and consonants for any input sentence specified in the URL.

Views.py

```
def vc(request,sentence):
    vow_cnt=0
    cons_cnt=0
    vow_dict=dict()
    cons_dict=dict()
    for letter in sentence:
        if letter.isalpha():
            if letter in "aeiouAEIOU":
                vow_cnt=vow_cnt+1
                vow_dict[letter]=vow_dict.get(letter,0)+1
            else:
                cons_cnt=cons_cnt+1
                cons_dict[letter]=cons_dict.get(letter,0)+1

    result="<h1>%d Vowels and %d Consonants</h1>" % (vow_cnt,cons_cnt)
    result+="<h2>Vowel Counter</h2>"
    for key,value in vow_dict.items():
        result+="<p>%s:%d</p>"%(key,value)
    result+="<h2>Consonant Counter</h2>"
    for key,value in cons_dict.items():
        result+="<p>%s:%d</p>"%(key,value)
    return HttpResponse(result)
```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import create_table_of_squares,vc
urlpatterns = [

    path('cts/<int:s>/<int:n>', create_table_of_squares), path('vc/<str:sentence>', vc),

]
```

Output:

6 vowels and 8 consonants

Vowel counter

a: 5

e: 1

Consonant counter

b: 1

n: 3

d: 1

p: 2

l: 1

Develop a Django app that finds the mode of a given set of numbers specified in the URL

Views.py

```
def find_mode(request,listofnum):
    arr=listofnum.split(",") num_count=dict()
    for num in arr:
        num_count[num]=num_count.get(num,0)+1
    num_count=sorted(num_count.items(),key=lambda item:item[1])
    num_count.reverse()
    result="<p><span style=color:red>%s</span> appears <span style=background-
color:yellow>%s</span> times"% (num_count[0][0],num_count[0][1])
    return HttpResponse(result)
```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import create_table_of_squares,vc,find_mode

urlpatterns = [
    path('admin/', admin.site.urls), path('cts/<int:s>/<int:n>',
    create_table_of_squares), path('vc/<str:sentence>', vc),
    path('find_mode/<str:listofnum>', find_mode),

]
```


Output:



127.0.0.1:8000/fm/1,3,2,4,3,1,2,2,6,7

2 occurred 3 times

Module-2: Django Templates and Models

Laboratory Component:

1. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event

Views.py

```
from datetime import date
from django.http import HttpResponse from
django.shortcuts import render
from django.template import Context, Template

# Create your views here.
def showlist(request):
    fruits=["Mango","Apple","Bananan","Jackfruits"]
    student_names=["Tony","Mony","Sony","Bob"]
    return render(request, 'showlist.html', {"fruits":fruits,"student_names":student_names}
)
```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import showlist

urlpatterns = [
    path('admin/', admin.site.urls),
    path('showlist/', showlist),

]
```

Template HTML file (inside ap2/templates subfolder)


showlist.html

```
<html>
  <style type="text/css">
    #i1 { background-color: lightgreen;color:brown;display:table} #i2
      { background-color: black;color:yellow}
  </style>
  <body>
    <h1 id="i1">Unordered list of fruits</h1>
    <ul>
      {% for fruit in fruits %}
        <li>{{ fruit }}</li>
      {% endfor %}

    </ul>
    <h1 id="i2">Ordered list of Students</h1>
    <ol>
      {% for student in student_names %}
        <li>{{ student }}</li>
      {% endfor %}

    </ol>
  </body>
</html>
```

Output:



Unordered list of fruits

- Mango
- Apple
- Bananan
- Jackfruits

Ordered list of Students

1. Tony
2. Mony
3. Sony
4. Bob

Develop a Django app that displays list of subject codes and subject names of any semester in tabular format. Even rows should have a light green background color and subject names should be in all caps

Views.py

```
from datetime import date
from django.http import HttpResponse from
django.shortcuts import render
from django.template import Context, Template

def list_of_subjects(request):
    s1={"scode":"21CS51","sname":"cn"}
    s2={"scode":"21CS52","sname":"ATc"}
    s3={"scode":"21CS53","sname":"DbMS"}
    s4={"scode":"21AI54","sname":"PAI"}
    l=list()
    l=[s1,s2,s3,s4]
    return render(request,'list_of_subjects.html',{ "l":l})
```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import create_table_of_squares,vc,find_mode from
ap2.views import list_of_subjects
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('list_of_subjects/', list_of_subjects),
]
```

Template file: list_of_subjects.html

```
<html>
<body>
    <table border>
        <tr>
            <th>Subject Code</th>
            <th>Subject Name</th>
        </tr>
        {% for subject in l %}
        {% if forloop.counter|divisibleby:"2" %}
            <tr>
```

```

<td style="background-color: lightgreen;">{{ subject.scode }}</td>
<td style="background-color: lightgreen;">{{ subject.sname|upper
}} </td>
</tr>
{% else %}
<tr>
<td>{{ subject.scode }}</td>
<td>{{ subject.sname|upper }}</td>
</tr>
{% endif %}
{% endfor %}
</table>
</body>

```

Output:

← → ↻ ⓘ 127.0.0.1:8000/list_of_subjects/

Subject Code	Subject Name
21CS51	CN
21CS52	ATC
21CS53	DBMS
21AI54	PAI

2. Develop a layout.html with a suitable header (containing navigation menu) and footer with copyright and developer information. Inherit this layout.html and create 3 additional pages: contact us, About Us and Home page of any website.

Views.py

```
from datetime import date
from django.http import HttpResponse from
django.shortcuts import render
from django.template import Context, Template
def home(request):
    return render(request,'home.html')

def aboutus(request):
    return render(request,'aboutus.html')

def contactus(request):
    return render(request,'contactus.html')
```

URLS.py

```
from django.contrib import admin
from django.urls import path, re_path
from ap2.views import aboutus, home, contactus

urlpatterns = [
    path('admin/', admin.site.urls),
    path('aboutus/', aboutus),
    path('home/', home),
    path('contactus/', contactus),

]
```

Template files:

layout.html

```
<html>
  <title>{% block title %} {% endblock %} </title>
  <style type="text/css">
    nav { background-color: lightblue;padding:10px }
  </style>
  <body>
    <nav>
      <a href="/home/">Home</a>|
      <a href="/aboutus/">About Us</a>|
      <a href="/contactus/">Contact Us</a>|
    </nav>
    <section>
      {% block content %} {% endblock %}
    </section>
    <footer>
      <hr>
      &copy; ISE, Developed by SK, Inc.
    </footer>
  </body>
</html>
```

home.html

```
{% extends 'layout.html' %}
{% block title %}
Home
{% endblock %}
{% block content %}
<h2>This is the home page</h2>
{% endblock %}
```

aboutus.html

```
{% extends 'layout.html' %}
{% block title %}
About Us
{% endblock %}
{% block content %}
<h2>We are Django developers</h2>
{% endblock %}
```

contactus.html

```
{% extends 'layout.html' %}
```

```
{% block title %}
```

Contact us

```
{% endblock %}
```

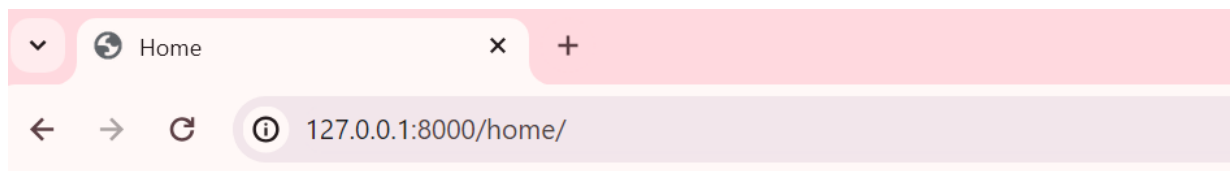
```
{% block content %}
```

<h2>Out phone: 9900923050
 Address:

K R Puram, Bangalore</h2>

```
{% endblock %}
```

Output:



This is the home page

© ISE @ CITEch, Developed by SK, Inc.

3. Develop a Django app that performs student registration to a course. It should also display list of students registered for any selected course. Create students and course as models with enrolment as ManyToMany field.

models.py

```
from django.db import models
```

```
class Course(models.Model):
    course_code = models.CharField(max_length=40)
    course_name = models.CharField(max_length=100)
    course_credits = models.IntegerField()
```

```
class Student(models.Model):
    student_usn = models.CharField(max_length=20)
    student_name = models.CharField(max_length=100)
    student_sem = models.IntegerField()
    enrolment = models.ManyToManyField(Course)
```

views.py

```
from django.shortcuts import render, HttpResponseRedirect
from .models import Student, Course
```

```
def reg(request):
    if request.method == "POST":
        sid = request.POST.get("sname")
        cid = request.POST.get("cname")
        student = Student.objects.get(id=sid)
        course = Course.objects.get(id=cid)
        res = student.enrolment.filter(id=cid)
        if res:
            return HttpResponseRedirect("<h1>Student already enrolled</h1>")
        student.enrolment.add(course)
        return HttpResponseRedirect("<h1>Student enrolled successfully</h1>")
    else:
        students = Student.objects.all()
        courses = Course.objects.all()
        return render(request, "reg.html", {"students": students, "courses":
courses})
```

```
def enrollment_list(request):
    students = Student.objects.all()
    enrollment_data = []
    for student in students:
        courses = student.enrolment.all()
```

```

        enrollment_data.append({
            'student_name': student.student_name,
            'courses': courses,
        })
    return render(request, "enrollment_list.html", {"enrollment_data":
enrollment_data})

```

In Application folder create forms.py file and include the below code

forms.py

```

from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        exclude = ['enrolment']

```

Under templates include two files reg.html and enrollment_list.html

reg.html

```

<!DOCTYPE html>
<html>
<body>
<form method="post" action="">
{% csrf_token %}
Student Name
<select name="sname">
    {% for student in students %}
        <option value="{ {{ student.id }}">{{ student.student_name }}</option>
    {% endfor %}
</select><br>
Course Name
<select name="cname">
    {% for course in courses %}
        <option value="{ {{ course.id }}">{{ course.course_name }}</option>
    {% endfor %}
</select><br>
<input type="submit" value="Enroll">
</form>
</body>
</html>

```

enrollment_list.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Enrollment List</title>
</head>
<body>
    <h1>Enrollment List</h1>
    <table border="1">
        <thead>
            <tr>
                <th>Student Name</th>
                <th>Enrolled Courses</th>
            </tr>
        </thead>
        <tbody>
            {% for enrollment in enrollment_data %}
                <tr>
                    <td>{{ enrollment.student_name }}</td>
                    <td>
                        <ul>
                            {% for course in enrollment.courses %}
                                <li>{{ course.course_name }} ({{course.course_code }})</li>
                            {% endfor %}
                        </ul>
                    </td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

admin.py

```
from django.contrib import admin
from .models import Student, Course
from .forms import StudentForm

class StudentAdmin(admin.ModelAdmin):
    form = StudentForm

admin.site.register(Student, StudentAdmin)
admin.site.register(Course)
```

urls.py

```
from django.contrib import admin
from django.urls import path

from ap1.views import enrollment_list, reg

urlpatterns = [
    path('admin/', admin.site.urls),
    path('reg/', reg),
    path('enrollment-list/', enrollment_list, name='enrollment_list'),
]
```

Screenshots

Backend – Adding Student details (*insert students through Admin interface)

The screenshot shows the Django Admin interface for adding a student. The browser address bar shows the URL `127.0.0.1:8000/admin/ap1/student/add/`. The page title is "Django administration" and the user is "SUNIL". The breadcrumb trail is "Home > Ap1 > Students > Add student". On the left sidebar, the "Students" link is highlighted. The main content area is titled "Add student" and contains three form fields: "Student usn:", "Student name:", and "Student sem:". At the bottom, there are three buttons: "SAVE", "Save and add another", and "Save and continue editing".

Course – Adding Course details (*insert courses through Admin interface)

The screenshot shows the Django Admin interface for adding a course. The browser address bar shows the URL `127.0.0.1:8000/admin/ap1/course/add/`. The page title is "Django administration" and the user is "SUNIL". The breadcrumb trail is "Home > Ap1 > Courses > Add course". On the left sidebar, the "Courses" link is highlighted. The main content area is titled "Add course" and contains three form fields: "Course code:", "Course name:", and "Course credits:". At the bottom, there are three buttons: "SAVE", "Save and add another", and "Save and continue editing".

Frontend – Registration

←

→

↺

🔒 127.0.0.1:8000/reg/

🔍 ☆ ⬇️ Ⓢ ⋮

Student Name

sunil ▾

Course Name

datascience ▾

Enroll

←

→

↺

🔒 127.0.0.1:8000/reg/

🔍 ☆ ⬇️ Ⓢ ⋮

Student enrolled successfully

←

→

↺

🔒 127.0.0.1:8000/reg/

🔍 ☆ ⬇️ Ⓢ ⋮

Student already enrolled

Front end - Enrollment List

←

→

↺

🔒 127.0.0.1:8000/enrollment-list/

🔍 ☆ ⬇️ Ⓢ ⋮

Enrollment List

Student Name	Enrolled Courses
SUNIL	<ul style="list-style-type: none">Django (01)
SANTHOSH	<ul style="list-style-type: none">Soil Conservation (02)

Module-3: Django Admin Interfaces and Model Forms

Laboratory Component:

1. For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.
2. Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.

1. For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.

models.py

```
from django.db import models

# Create your models here.
class Course(models.Model):
    course_code=models.CharField(max_length=40)
    course_name=models.CharField(max_length=100)
    course_credits=models.IntegerField(blank=True, null=True)
    def __str__(self):
        return self.course_name

class Student(models.Model):
    student_usn=models.CharField(max_length=20)
    student_name=models.CharField(max_length=100)
    student_sem=models.IntegerField()
    enrolment=models.ManyToManyField(Course)
    def __str__(self):
        return self.student_name+"("+self.student_usn+")"
```

admin.py

```
from django.contrib import admin

# Register your models here.
from ap1.models import Course, Student
# Register your models here.
#admin.site.register(Student)
admin.site.register(Course)

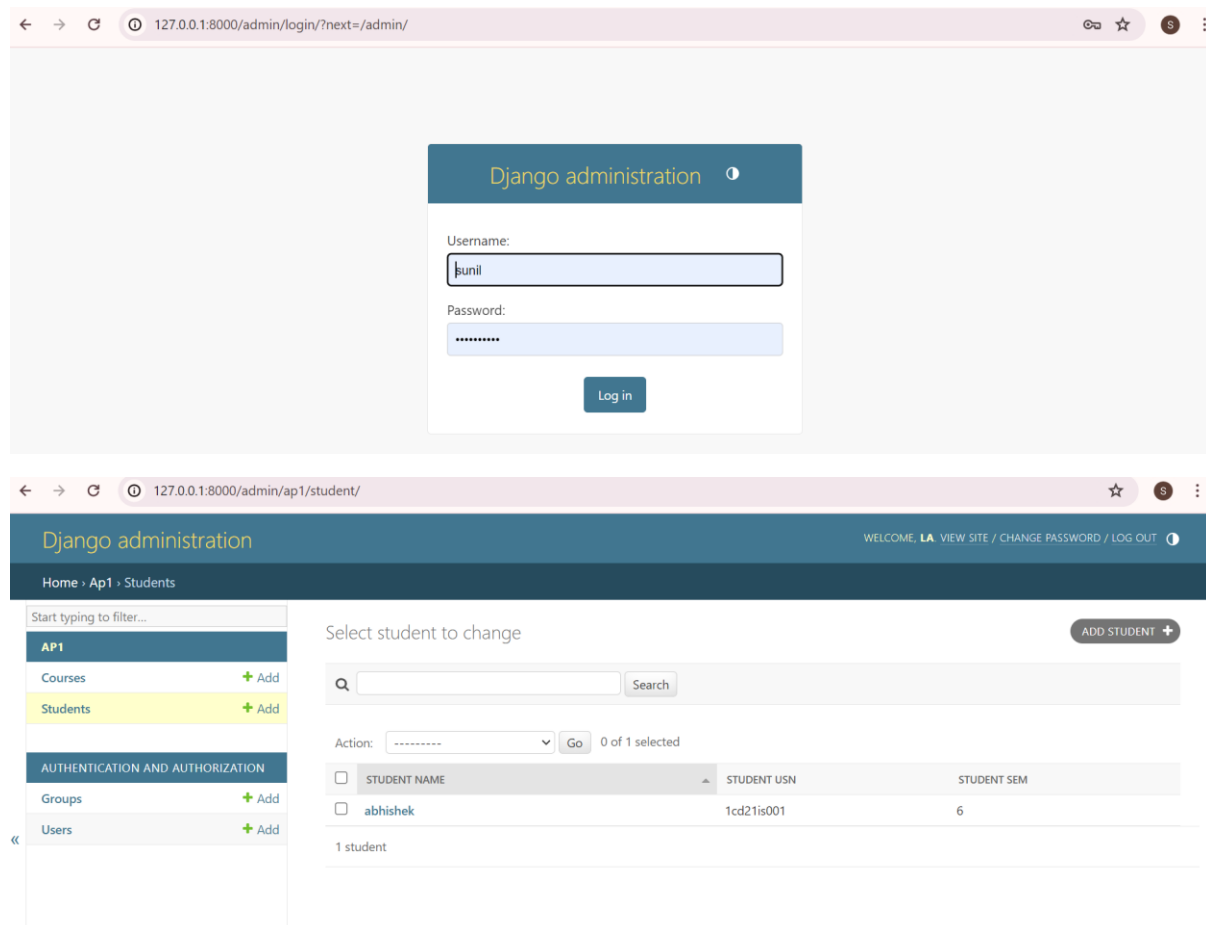
class StudentAdmin(admin.ModelAdmin):
    list_display = ('student_name', 'student_usn', 'student_sem')
    ordering= ('student_name',)
    search_fields = ('student_name',)

admin.site.register(Student, StudentAdmin)
```

Next follow these steps

- ❖ Open the settings.py file in your project.
- ❖ Find the list called INSTALLED_APPS.
- ❖ Add 'ap1' to the INSTALLED_APPS list.
- ❖ Save and close the settings.py file.
- ❖ Open your command line or terminal.
- ❖ Type `python manage.py makemigrations` and press Enter.
- ❖ Then type `python manage.py migrate` and press Enter.
- ❖ Type `python manage.py createsuperuser` and press Enter.
- ❖ Follow the prompts to enter a username, email address, and password.
- ❖ After creating the superuser, start the development server by typing `python manage.py runserver` and pressing Enter.
- ❖ Open your web browser and go to `http://127.0.0.1:8000/admin/`.
- ❖ Log in using the superuser credentials you created.
- ❖ Once logged in, you can add, edit, or delete data from the admin interface.

Output :



Django administration

WELCOME, LA VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Ap1 > Students > Add student

Start typing to filter...

AP1

Courses + Add

Students + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

Add student

Student usn:

Student name:

Student sem:

Enrolment:

sep
django

Hold down "Control", or "Command" on a Mac, to select more than one.

SAVE

Save and add another

Save and continue editing

Django administration

WELCOME, LA VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Ap1 > Courses > Add course

Start typing to filter...

AP1

Courses + Add

Students + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

Add course

Course code:

Course name:

Course credits:

SAVE

Save and add another

Save and continue editing

Django administration

WELCOME, LA VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Ap1 > Students

Start typing to filter...

AP1

Courses + Add

Students + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

Select student to change

ADD STUDENT +

Q

Search

Action: ----- Go 0 of 1 selected

<input type="checkbox"/>	STUDENT NAME	STUDENT USN	STUDENT SEM
<input type="checkbox"/>	abhishek	1cd21is001	6

1 student

2. Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.

models.py

```
from django.db import models
from django.forms import ModelForm

# Create your models here.
class Course(models.Model):
    course_code=models.CharField(max_length=40)
    course_name=models.CharField(max_length=100)
    course_credits=models.IntegerField(blank=True, null=True)
    def __str__(self):
        return self.course_name

class Student(models.Model):
    student_usn=models.CharField(max_length=20)
    student_name=models.CharField(max_length=100)
    student_sem=models.IntegerField()
    enrolment=models.ManyToManyField(Course)
    def __str__(self):
        return self.student_name+"("+self.student_usn+")"

class Project(models.Model):
    student=models.ForeignKey(Student,on_delete=models.CASCADE)
    ptopic=models.CharField(max_length=200)
    plangauges=models.CharField(max_length=200)
    pduration=models.IntegerField()

class ProjectReg(ModelForm):
    required_css_class="required"
    class Meta:
        model=Project
        fields=['student','ptopic','plangauges','pduration']
```

views.py

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse
from django.shortcuts import render
from ap1.models import ProjectReg
def add_project(request):
    if request.method=="POST":
        form=ProjectReg(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponse("<h1>Record inserted successfully</h1>")
```

```

else:
    return HttpResponseRedirect("<h1>Record not inserted</h1>")
else:
    form=ProjectReg()
    return render(request,"add_project.html",{ "form":form})

```

- ❖ In your application folder, create a new folder named templates.
- ❖ Inside the templates folder, create a file named add_project.html.

add_project.html

```

<html>
<form method="post" action="">
    {% csrf_token %}
    <table>
        {{ form.as_table }}
    <tr>
        <td>
            <input type="submit" value="Submit">
        </td>
    </tr>
    </table>
</form>
</html>

```

urls.py

```

from django.contrib import admin
from django.urls import path

from ap1.views import add_project

urlpatterns = [
    path('admin/', admin.site.urls),
    path('add_project/', add_project),
]


```

Next follow these steps

- ❖ Open the settings.py file in your project.
- ❖ Find the list called INSTALLED_APPS.
- ❖ Add 'ap1' to the INSTALLED_APPS list.
- ❖ Save and close the settings.py file.
- ❖ Open your command line or terminal.
- ❖ Type `python manage.py makemigrations` and press Enter.
- ❖ Then type `python manage.py migrate` and press Enter.
- ❖ Type `python manage.py createsuperuser` and press Enter.
- ❖ Follow the prompts to enter a username, email address, and password.
- ❖ After creating the superuser, start the development server by typing `python manage.py runserver` and pressing Enter.
- ❖ Open your web browser and go to http://127.0.0.1:8000/add_project/

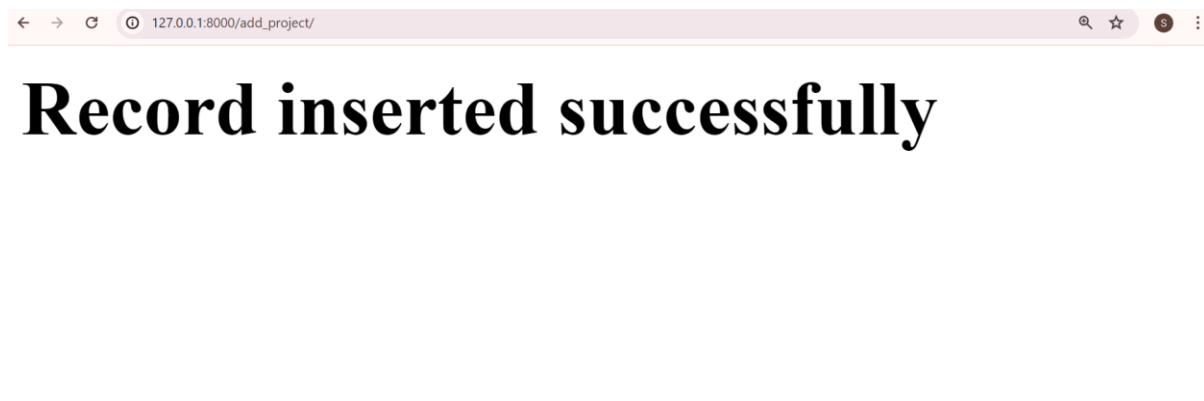
- ❖ Fill in the details in the form.
- ❖ Click the "Submit" button.
- ❖ The record will be inserted into the database.

Output



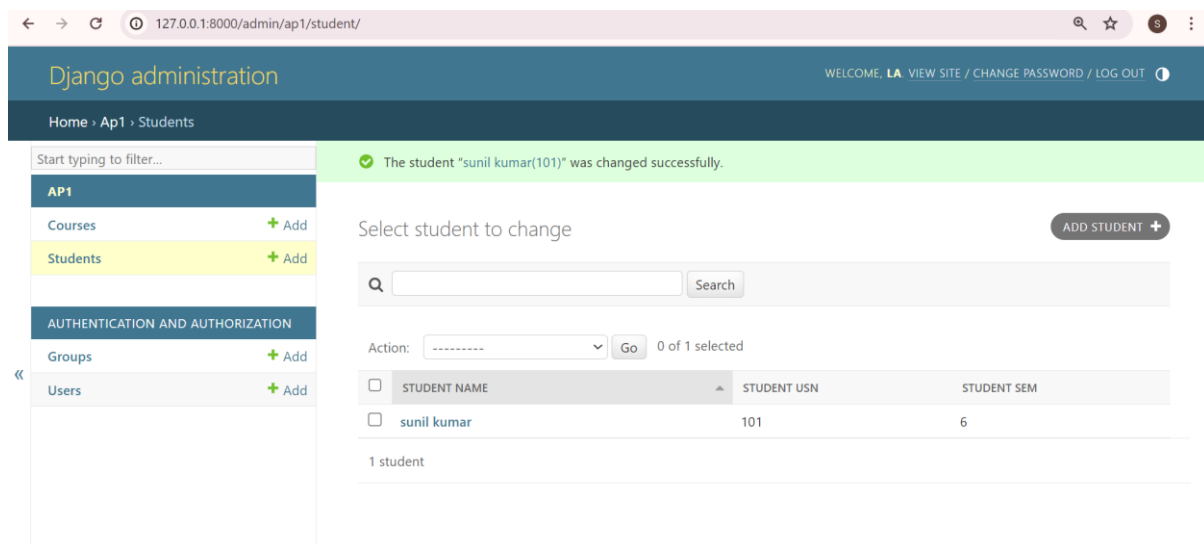
A web browser window showing a form at the URL 127.0.0.1:8000/add_project/. The form contains the following fields:

- Student:** A dropdown menu with a placeholder text "-----".
- Ptopic:** A text input field.
- Plangauges:** A text input field.
- Pduration:** A text input field.
- Submit:** A button to submit the form.



A web browser window showing a confirmation message at the URL 127.0.0.1:8000/add_project/. The message is:

Record inserted successfully



A screenshot of the Django administration interface. The top navigation bar shows "Django administration" and "WELCOME LA VIEW SITE / CHANGE PASSWORD / LOG OUT". The left sidebar shows the navigation menu with "AP1" selected. The main content area shows a message: "The student 'sunil kumar(101)' was changed successfully." Below this, there is a section titled "Select student to change" with a search bar and a table of students.

STUDENT NAME	STUDENT USN	STUDENT SEM
<input type="checkbox"/> sunil kumar	101	6

1 student