Maratha Mandal’s Engineering College, Belagavi

Department of Computer Science & Engineering

Fullstack Development Lab Manual(21CS62)

Module 1

1. Installation of Python, Django and Visual Studio code editors can be demonstrated.
Here’s a step-by-step guide to

Step 1: Install Python

Ubuntu typically comes with Python pre-installed. To check the installed version and install Python3 if it’s not already available, follow these commands:

Update package lists:

```
sudo apt update
```

Check if Python is installed:

```
python3 --version
```

If Python 3 is not installed, install it:

```
sudo apt install python3
```

Step 2: Install pip (Python Package Installer)

First, update `pip` to the latest version:

```
sudo apt install python3-pip
```

Step 3: Install Django

Use `pip` to install Django:

```
pip3 install django
```

Verify the installation:

```
django-admin --version
```

Step 4: Install Visual Studio Code

1. Download the `.deb` package for Visual Studio Code from the official website: Download Visual Studio Code for Linux.
2. Once the download is complete, navigate to the directory containing the `.deb` file in the terminal.
3. Install Visual Studio Code using `dpkg`:

```
sudo dpkg -i <package-file-name>.deb
```

Replace `<package-file-name>` with the name of the `.deb` package you downloaded.

Resolve dependencies, if any:

```
sudo apt install -f
```

Step 5: Launch Visual Studio Code

You can launch Visual Studio Code from the applications menu, or by running `code` from the terminal.

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

Here are the detailed steps to create a virtual environment, set up a Django project, and create a Django app on Ubuntu.

Step 1: Install Virtualenv

```
pip3 install virtualenv
```

Step 2: Create a Virtual Environment

Navigate to the directory where you want to create your project:

```
cd /path/to/your/project
```

Create a virtual environment:

```
$ virtualenv venv
```

This will create a directory named `venv` containing the virtual environment.

Activate the virtual environment:

```
source venv/bin/activate
```

You should see `(venv)` preceding your command prompt, indicating that the virtual environment is active.

Step 3: Install Django in the Virtual Environment

```
$ pip install django
Collecting django
  Downloading Django-5.0.6-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.7.0 (from django)
  Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from django)
  Downloading sqlparse-0.5.0-py3-none-any.whl.metadata (3.9 kB)
Downloading Django-5.0.6-py3-none-any.whl (8.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 17.4 MB/s eta 0:00:00
Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
Downloading sqlparse-0.5.0-py3-none-any.whl (43 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.0/44.0 kB 599.5 kB/s eta 0:00:00
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.0.6 sqlparse-0.5.0
```

Step 4: Create a Django Project

```
django-admin startproject myproject
```

This will create a directory named `myproject` with the following structure:

```
myproject/
    manage.py
    myproject/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Navigate into the project directory:

```
cd myproject
```

Step 5: Create a Django App

```
python manage.py startapp myapp
```

This will create a directory named `myapp` with the following structure:

```
myapp/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Step 6: Add the App to the Project

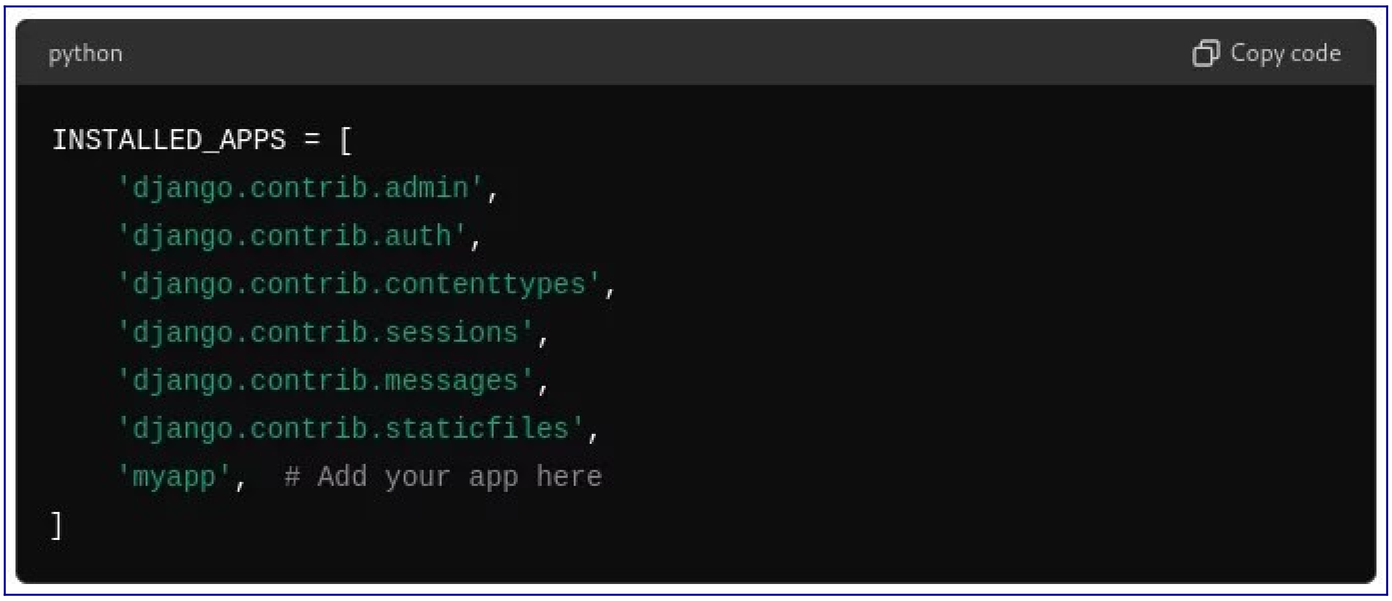
Open myproject/settings.py in a text editor (e.g., Visual Studio Code):

```
code myproject/settings.py
```

Add `myapp` to the `INSTALLED_APPS` list:

```
INSTALLED_APPS = [
    ...
    'myapp',
]
```

After adding it may appear like this



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',  # Add your app here
]
```

Step 7: Run the Development Server

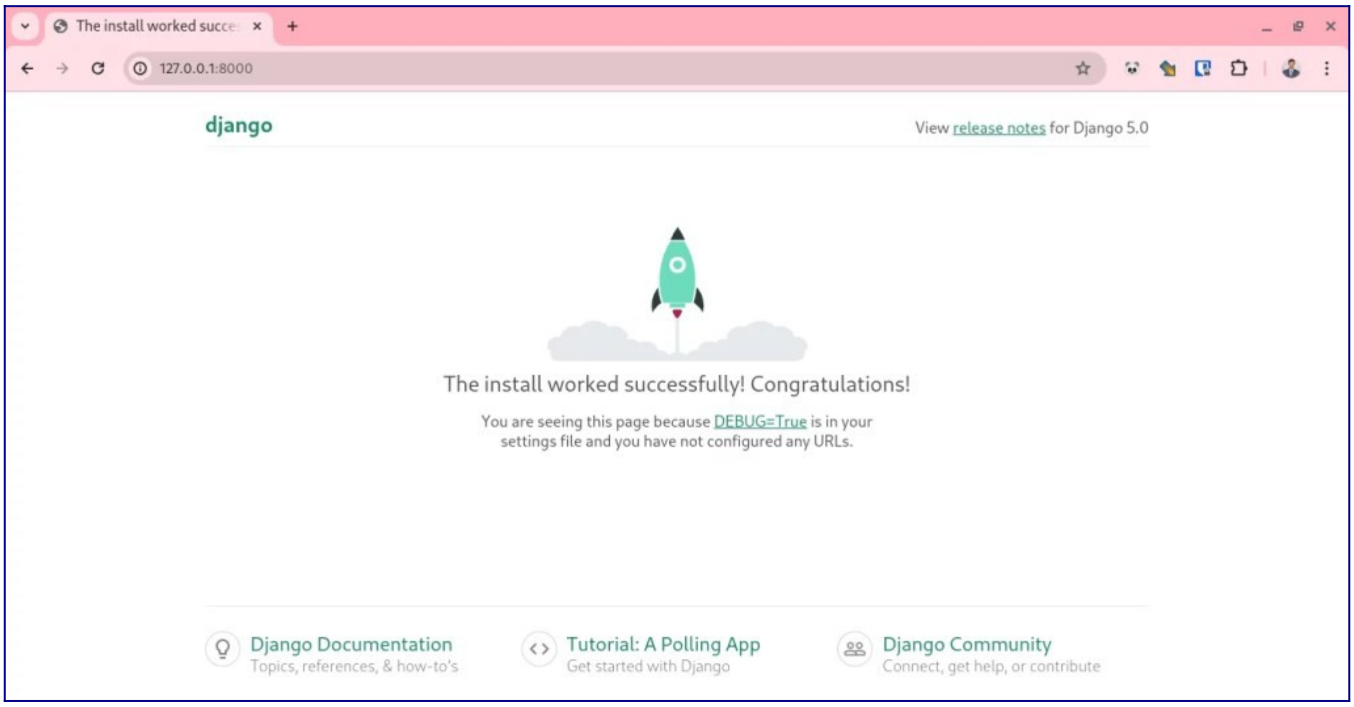
Apply migrations:

```
putta:~/.../myproject$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Run the development server:

```
python manage.py runserver
```

Open your web browser and navigate to `http://127.0.0.1:8000/` to see the default Django welcome page.



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Summary

You've now successfully:

- Created a virtual environment.
- Installed Django within the virtual environment.

- Created a new Django project named `myproject`.
- Created a new Django app named `myapp`.
- Configured the project to include the new app.

- Run the Django development server to test the setup.

You can now start developing your Django application by adding models, views, and templates within your `myapp` directory.

---

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

Let's create a Django app that displays the current date and time on a web page. We'll go through the following steps:

- **Set up the Django project and app.**
- **Create a view to display the current date and time.**
- **Configure URLs.**
- **Create a template to display the date and time.**
- **Run the server and test the application.**

Step-by-Step Guide

Step 1: Set Up the Django Project and App

```
django-admin startproject mydatetime
cd mydatetime
```

Create a Django app named `datetime_display`:

```
python manage.py startapp datetime_display
```

Add `datetime_display` to `INSTALLED_APPS` in `mydatetime/settings.py`:

```
# mydatetime/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'datetime_display',  # Add your app here
]
```

Also in the `settings.py` file of your project, set the `TIME_ZONE` variable to 'Asia/Kolkata' (which corresponds to IST):

```
# mydatetime/settings.py

TIME_ZONE = 'Asia/Kolkata'
USE_TZ = True
```

Step 2: Create a View to Display the Current Date and Time

Edit the `views.py` file in the `datetime_display` app:

```
# datetime_display/views.py

from django.shortcuts import render
from django.http import HttpResponse
from datetime import datetime

def current_datetime(request):
    now = datetime.now()
    context = {
        'current_date': now,
    }
    return render(request, 'datetime_display/current_datetime.html', context)
```

Step 3: Configure URLs

Create a `urls.py` file in the `datetime_display` app:

```
# datetime_display/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('current_datetime/', views.current_datetime, name='current_datetime'),
]
```

Include the `datetime_display` URLs in the main `urls.py` file:

```
# myproject/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('datetime_display.urls')),  # Include the app URLs
]
```

Step 4: Create a Template to Display the Date and Time

Create a directory named `templates` in the `datetime_display` app:

```
mkdir templates
```

Create a template file `current_datetime.html` in the `mydatetime/templates/` directory

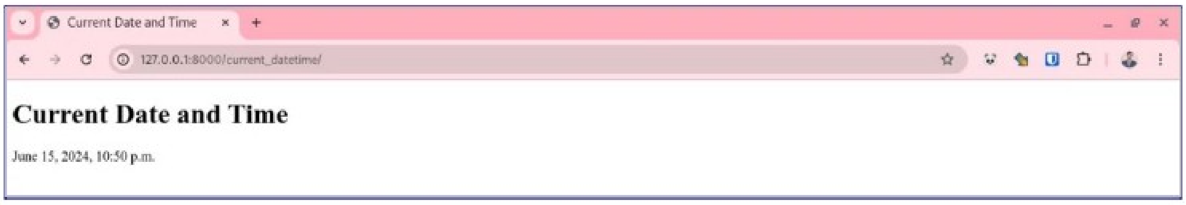
```
<!-- mydatetime/templates/current_datetime.html -->
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Current Date and Time</title>
</head>
<body>
    <h1>Current Date and Time</h1>
    <p>{{ current_date }}</p>
</body>
</html>
```

Step 5: Run the Server and Test the Application

```
python manage.py runserver
```

Open a web browser and navigate to `http://127.0.0.1:8000/current_datetime/` to see the current date and time displayed on the web page.



Current Date and Time

June 15, 2024, 10:50 p.m.

Summary

You’ve now created a Django app that displays the current date and time. Here’s a summary of what we did:

- Created a Django project and an app named `datetime_display`.
- Added the app to the `INSTALLED_APPS` list.
- Created a view to get the current date and time.
- Configured the URLs to route to the view.
- Created a template to display the date and time.
- Ran the server and tested the application.

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

Django app that displays the current date and time, along with the date and time four hours ahead and four hours before the current date and time.

Step 1: Set Up the Django Project and App Create a Django project (if not already created):

```
django-admin startproject timeoffset
cd timeoffset
```

Create a Django app named `time_offset`:

```
python manage.py startapp time_offset
```

Add `time_offset` to `INSTALLED_APPS` in `myproject/settings.py` and set the time zone:

```
# timeoffset/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'time_offset',  # Add your app here
]

# Set the timezone
TIME_ZONE = 'Asia/Kolkata'
USE_TZ = True
```

Step 2: Create Views to Display the Date and Time

Edit the `views.py` file in the `time_offset` app:

```
# time_offset/views.py

from django.shortcuts import render
from django.utils import timezone
from datetime import timedelta

def current_datetime(request):
    now = timezone.now()
    four_hours_ahead = now + timedelta(hours=4)
    four_hours_before = now - timedelta(hours=4)
    context = {
        'current_date': now,
        'four_hours_ahead': four_hours_ahead,
        'four_hours_before': four_hours_before,
    }
    return render(request, 'time_offset/current_datetime.html', context)
```

Step 3: Configure URLs

Create a `urls.py` file in the `time_offset` app:

```
# time_offset/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('time_offsets/', views.current_datetime, name='current_datetime'),
]
```

Include the `time_offset` URLs in the main `urls.py` file:

```
# timeoffset/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('time_offset.urls')),  # Include the app URLs
]
```

Step 4: Create a Template to Display the Dates and Times

Create a directory named `templates` in the `time_offset` app:

```
mkdir -p time_offset/templates/time_offset
```

Create a template file `current_datetime.html` in the `time_offset/templates/time_offset` directory:

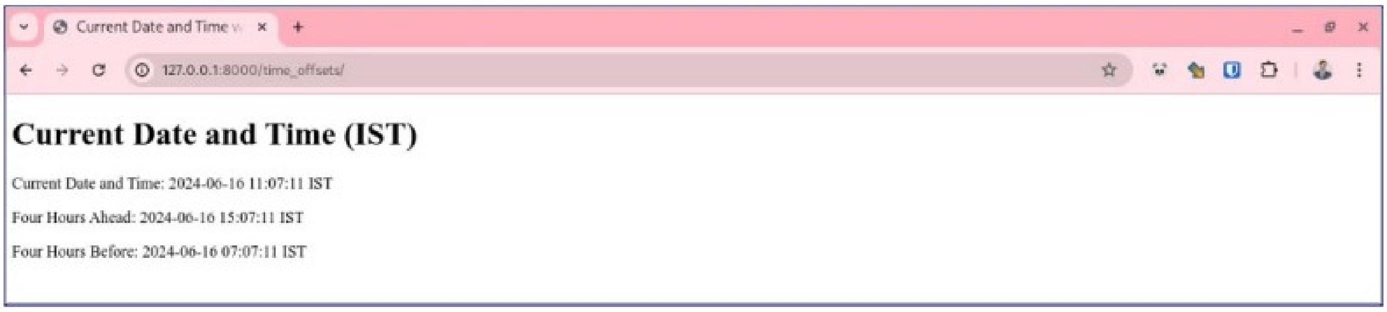
```
<!-- time_offset/templates/time_offset/current_datetime.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Current Date and Time with Offsets</title>
</head>
<body>
    <h1>Current Date and Time (IST)</h1>
    <p>Current Date and Time: {{ current_date|date:"Y-m-d H:i:s T" }}</p>
    <p>Four Hours Ahead: {{ four_hours_ahead|date:"Y-m-d H:i:s T" }}</p>
    <p>Four Hours Before: {{ four_hours_before|date:"Y-m-d H:i:s T" }}</p>
</body>
</html>
```

Step 5: Run the Server and Test the Application

Run the Django development server:

```
python manage.py runserver
```

Open a web browser and navigate to `http://127.0.0.1:8000/time_offsets/` to see the current date and time displayed on the web page.



Current Date and Time (IST)

Current Date and Time: 2024-06-16 11:07:11 IST

Four Hours Ahead: 2024-06-16 15:07:11 IST

Four Hours Before: 2024-06-16 07:07:11 IST

You've now created a Django app that displays:

- The current date and time.
- The date and time four hours ahead.
- The date and time four hours before.

Here's a summary of what we did:

- Created a Django project and an app named `time_offset`.
- Added the app to the `INSTALLED_APPS` list.
- Created a view to calculate the required times.
- Configured the URLs to route to the view.
- Created a template to display the date and time with the offsets.
- Ran the server and tested the application.

Module 2

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

Let's develop a Django app that displays an unordered list of fruits and an ordered list of selected students for a programming contest.

Step 1: Set Up the Django Project and App

Create a Django project (if not already created):

```
django-admin startproject mylist
cd mylist/
```

Create a Django app named `lists_display`:

```
python manage.py startapp lists_display
```

Add `lists_display` to `INSTALLED_APPS` in `mylist/settings.py`:

```
# mylist/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'lists_display',  # Add your app here
]
```

Step 2: Create Views to Display the Lists

Edit the `views.py` file in the `lists_display` app:

```
# lists_display/views.py

from django.shortcuts import render

def display_lists(request):
    fruits = ['Apple', 'Banana', 'Mango', 'Jackfruit', 'Grapes']
    students = ['Jamal', 'Suresh', 'Charlie', 'Divya', 'Raja']
    context = {
        'fruits': fruits,
        'students': students,
    }
    return render(request, 'lists_display/display_lists.html', context)
```

Step 3: Configure URLs

Create a `urls.py` file in the `lists_display` app:

```
# lists_display/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('display_lists/', views.display_lists, name='display_lists'),
]
```

Include the `lists_display` URLs in the main `urls.py` file:

```
# mylist/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('lists_display.urls')),  # Include the app URLs
]
```

Step 4: Create a Template to Display the Lists

Create a directory named `templates` in the `lists_display` app:

```
mkdir templates
```

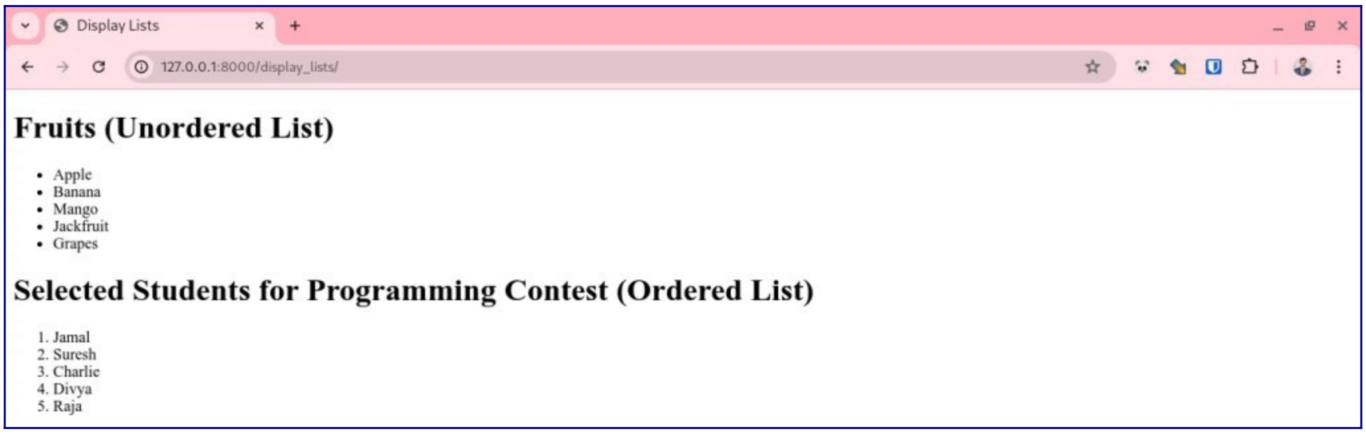
Create a template file `display_lists.html` in the `templates` directory:

```
<!-- lists_display/templates/lists_display/display_lists.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Display Lists</title>
</head>
<body>
    <h1>Fruits (Unordered List)</h1>
    <ul>
        {% for fruit in fruits %}
            <li>{{ fruit }}</li>
        {% endfor %}
    </ul>

    <h1>Selected Students for Programming Contest (Ordered List)</h1>
    <ol>
        {% for student in students %}
            <li>{{ student }}</li>
        {% endfor %}
    </ol>
</body>
</html>
```

Step 5: Run the Server and Test the Application

```
python manage.py runserver
```

Open a web browser and navigate to `http://127.0.0.1:8000/display_lists/` to see the current date and time displayed on the web page.


Display Lists

127.0.0.1:8000/display_lists/

Fruits (Unordered List)

- Apple
- Banana
- Mango
- Jackfruit
- Grapes

Selected Students for Programming Contest (Ordered List)

1. Jamal
2. Suresh
3. Charlie
4. Divya
5. Raja

You've now created a Django app that displays:

- An unordered list of fruits.
- An ordered list of selected students for a programming contest.

Here's a summary of what we did:

- Created a Django project and an app named `lists_display`.
- Added the app to the `INSTALLED_APPS` list.
- Created a view to provide the lists.
- Configured the URLs to route to the view.
- Created a template to display the lists.
- Ran the server and tested the application.

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.

Let's create a Django app with a base layout that includes a header with a navigation menu and a footer. Then, we'll create three additional pages: "Contact Us," "About Us," and "Home" for a college website, all of which will inherit from this base layout.

Step 1: Set Up the Django Project and App

Create a Django project (if not already created):

```
django-admin startproject mycollege
cd mycollege
```

Create a Django app named `website`:

```
python manage.py startapp website
```

Add `website` to `INSTALLED_APPS` in `mycollege/settings.py`:

```
# mycollege/settings.py

import os

# Static files (CSS, JavaScript, Images)
STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'website',  # Add your app here
]
```

Step 2: Create the Base Layout Template

Create directories for templates:

```
mkdir templates
```

Create `layout.html` template in the `templates` directory:

```
<!-- website/templates/website/layout.html -->

<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}College Website{% endblock %}</title>
</head>
<body>
    <header>
        <h1>Welcome to Our College</h1>
        <nav>
            <ul>
                <li><a href="{% url 'home' %}">Home</a></li>
                <li><a href="{% url 'about' %}">About Us</a></li>
                <li><a href="{% url 'contact' %}">Contact Us</a></li>
            </ul>
        </nav>
    </header>

    <main>
        {% block content %}{% endblock %}
    </main>

    <footer>
        <p>© 2024 My College. All rights reserved.</p>
        <p>Developed by Your Name</p>
    </footer>
</body>
</html>
```

Step 3: Create the Additional Pages

Create `home.html` template:

```
<!-- website/templates/website/home.html -->

{% extends 'website/layout.html' %}
{% load static %}

{% block title %}Home{% endblock %}

{% block content %}
<h2>Home Page</h2>
<p>Welcome to the home page of our college website.</p>
<img src="{% static 'images/college.jpg' %}" alt="MMEC" width="250" height="150">
{% endblock %}
```

Create a `static` directory in your project root if it does not already exist, Create an `images` directory within the `static` directory:

```
mkdir -p static/images
```

Add your image file to the `static/images` directory. For this example, let's assume your image is named `college.jpg`

Create `about.html` template:

```
<!-- website/templates/website/about.html -->

{% extends 'website/layout.html' %}

{% block title %}About Us{% endblock %}

{% block content %}
<h2>About Us</h2>
<p>Learn more about our college, our history, and our mission.</p>
{% endblock %}
```

Create `contact.html` template:

```
<!-- website/templates/website/contact.html -->

{% extends 'website/layout.html' %}

{% block title %}Contact Us{% endblock %}

{% block content %}
<h2>Contact Us</h2>
<p>Get in touch with us for more information.</p>
{% endblock %}
```

Step 4: Create Views for Each Page

Edit the `views.py` file in the `website` app:

```
# website/views.py

from django.shortcuts import render

def home(request):
    return render(request, 'website/home.html')

def about(request):
    return render(request, 'website/about.html')

def contact(request):
    return render(request, 'website/contact.html')
```

Step 5: Configure URLs

Create a `urls.py` file in the `website` app:

```
# website/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Include the `website` URLs in the main `urls.py` file:

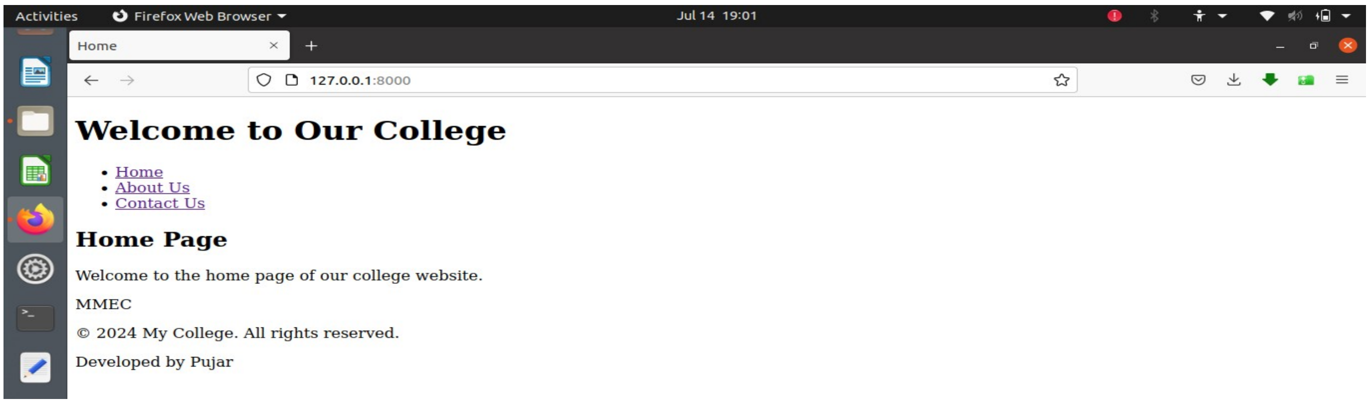
```
# mycollege/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('website.urls')),  # Include the app URLs
]
```

Step 6: Run the Server and Test the Application

```
python manage.py runserver
```

Open a web browser and navigate to:

- `http://127.0.0.1:8000/` for the Home page.
- `http://127.0.0.1:8000/about/` for the About Us page.
- `http://127.0.0.1:8000/contact/` for the Contact Us page.



Welcome to Our College

- Home
- About Us
- Contact Us

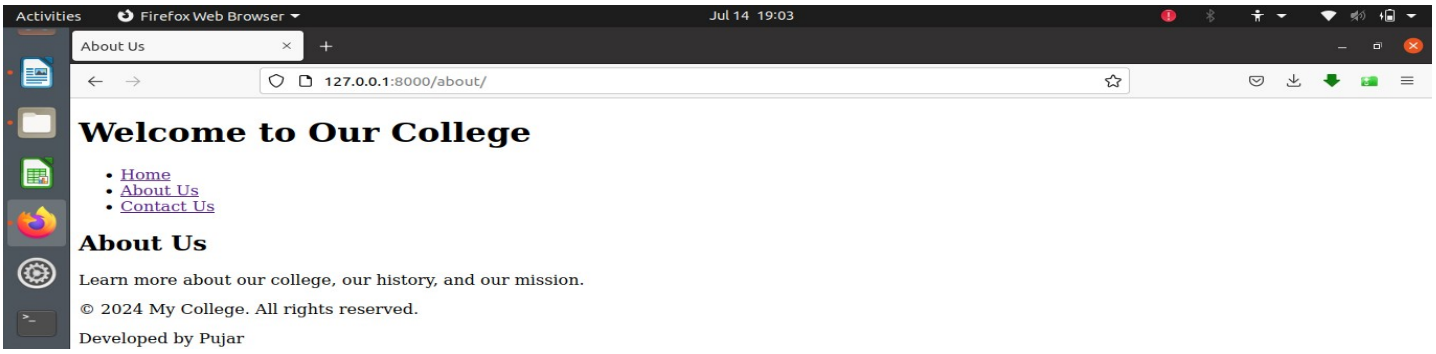
Home Page

Welcome to the home page of our college website.

MMEC

© 2024 My College. All rights reserved.

Developed by Pujar



Welcome to Our College

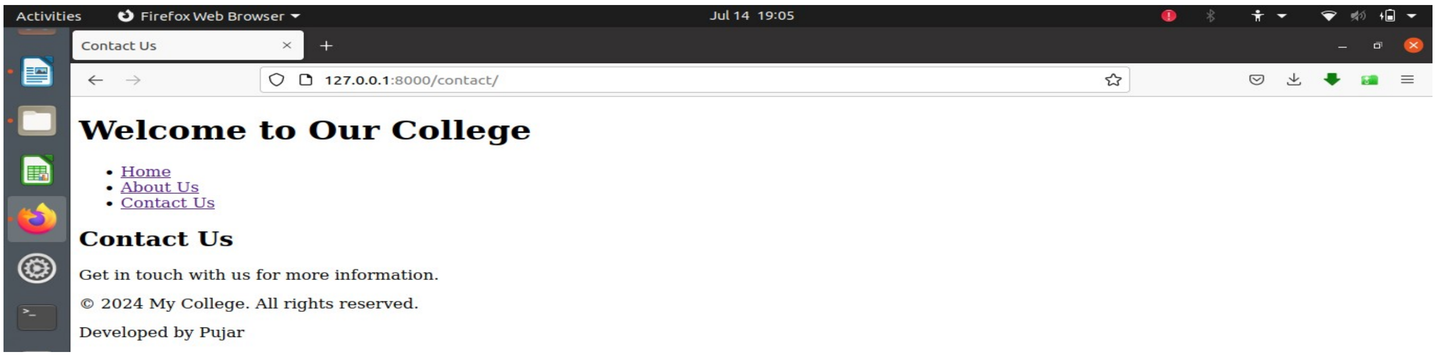
- Home
- About Us
- Contact Us

About Us

Learn more about our college, our history, and our mission.

© 2024 My College. All rights reserved.

Developed by Pujar



Welcome to Our College

- Home
- About Us
- Contact Us

Contact Us

Get in touch with us for more information.

© 2024 My College. All rights reserved.

Developed by Pujar

3. Course Registration **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.**
To develop a Django app that handles student registration to courses and displays a list of students registered for any selected course, you can follow these steps:

Step 1: Set Up the Django Project and App

Create a Django project (if not already created):

```
django-admin startproject mycollege
cd mycollege
```

Create a Django app named `courses`:

```
python manage.py startapp courses
```

Add courses to INSTALLED_APPS in mycollege/settings.py:

```
# mycollege/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'courses',  # Add your app here
]
```

Step 2: Create Models

Edit `models.py` in the `courses` app to create `Student` and `Course` models with a ManyToMany relationship:

```
# courses/models.py

from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

class Course(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    students = models.ManyToManyField(Student, related_name='courses')

    def __str__(self):
        return self.name
```

Create and apply the migrations:

```
python manage.py makemigrations
python manage.py migrate
```

Step 3: Create Admin Interface

Register the models in the admin site by editing `admin.py`:

```
# courses/admin.py

from django.contrib import admin
from .models import Student, Course

```

```
class StudentAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')
    search_fields = ('first_name', 'last_name', 'email')

class CourseAdmin(admin.ModelAdmin):
    list_display = ('name', 'description')
    search_fields = ('name',)
    filter_horizontal = ('students',)

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

Step 4: Create Forms for Registration

Create a `forms.py` file in the `courses` app to handle the registration form:

```
# courses/forms.py

from django import forms
from .models import Student, Course

class StudentRegistrationForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['first_name', 'last_name', 'email']

class CourseSelectionForm(forms.Form):
    course = forms.ModelChoiceField(queryset=Course.objects.all(), required=True,
label="Select a Course")
```

Step 5: Create Views for Registration and Displaying Students

Edit `views.py` to create views for student registration and displaying students by course:

```
# courses/views.py

from django.shortcuts import render, redirect, get_object_or_404
from .models import Student, Course
from .forms import StudentRegistrationForm, CourseSelectionForm

def register_student(request):
    if request.method == 'POST':
        form = StudentRegistrationForm(request.POST)
        if form.is_valid():
            student = form.save()
            return redirect('select_course', student_id=student.id)
    else:
        form = StudentRegistrationForm()
    return render(request, 'courses/register_student.html', {'form': form})

def select_course(request, student_id):
    student = get_object_or_404(Student, id=student_id)
    if request.method == 'POST':
        form = CourseSelectionForm(request.POST)
        if form.is_valid():
            course = form.cleaned_data['course']
            course.students.add(student)
            return redirect('course_students', course_id=course.id)
    else:
        form = CourseSelectionForm()
    return render(request, 'courses/select_course.html', {'form': form, 'student':
student})

def course_students(request, course_id):
    course = get_object_or_404(Course, id=course_id)
    students = course.students.all()
    return render(request, 'courses/course_students.html', {'course': course, 'students':
students})
```

Step 6: Configure URLs

Create a `urls.py` file in the `courses` app:

```
# courses/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register_student, name='register_student'),
    path('select_course/<int:student_id>/', views.select_course, name='select_course'),
    path('course_students/<int:course_id>/', views.course_students,
name='course_students'),
]
```

Include the `courses` URLs in the main `urls.py` file:

```
# mycollege/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('courses.urls')),  # Include the app URLs
]
```

Step 7: Create Templates

Create a directory named `templates` in the `courses` app:

```
mkdir -p courses/templates/courses
```

Create `register_student.html` template:

```
<!-- courses/templates/courses/register_student.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Register Student</title>
</head>
<body>
    <h1>Register Student</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Register</button>
    </form>
</body>
</html>
```

Create `select_course.html` template:

```
<!-- courses/templates/courses/select_course.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Select Course</title>
</head>
<body>
    <h1>Select Course for {{ student.first_name }} {{ student.last_name }}</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Enroll</button>
    </form>
</body>
</html>
```

Create `course_students.html` template:

```
<!-- courses/templates/courses/course_students.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Students Enrolled in {{ course.name }}</title>
</head>
<body>
    <h1>Students Enrolled in {{ course.name }}</h1>
    <ul>
        {% for student in students %}
            <li>{{ student.first_name }} {{ student.last_name }}</li>
        {% endfor %}
    </ul>
    <a href="{% url 'register_student' %}">Register another student</a>
</body>
</html>
```

Step 8: Add Initial Courses

Create a Django management command to add initial courses:

Create a directory named `management/commands` inside the `courses` app:

```
mkdir -p courses/management/commands
```

Create a new file named `add_initial_courses.py` inside the `commands` directory:

```
touch courses/management/commands/add_initial_courses.py
```

Edit the `add_initial_courses.py` file to add the initial courses:

```
# courses/management/commands/add_initial_courses.py

from django.core.management.base import BaseCommand
from courses.models import Course

class Command(BaseCommand):
    help = 'Add initial courses to the database'

    def handle(self, *args, **kwargs):
        courses = [
            {'name': 'Mathematics', 'description': 'Basic Mathematics'},
            {'name': 'Physics', 'description': 'Fundamentals of Physics'},
            {'name': 'Chemistry', 'description': 'Basics of Chemistry'},
            {'name': 'Biology', 'description': 'Introduction to Biology'},
            {'name': 'Computer Science', 'description': 'Basics of Computer Science'},
        ]

        for course in courses:
            Course.objects.get_or_create(name=course['name'], defaults={'description':
course['description']})

        self.stdout.write(self.style.SUCCESS('Successfully added initial courses'))
```

Run the custom management command to add initial courses:

```
python manage.py add_initial_courses
```

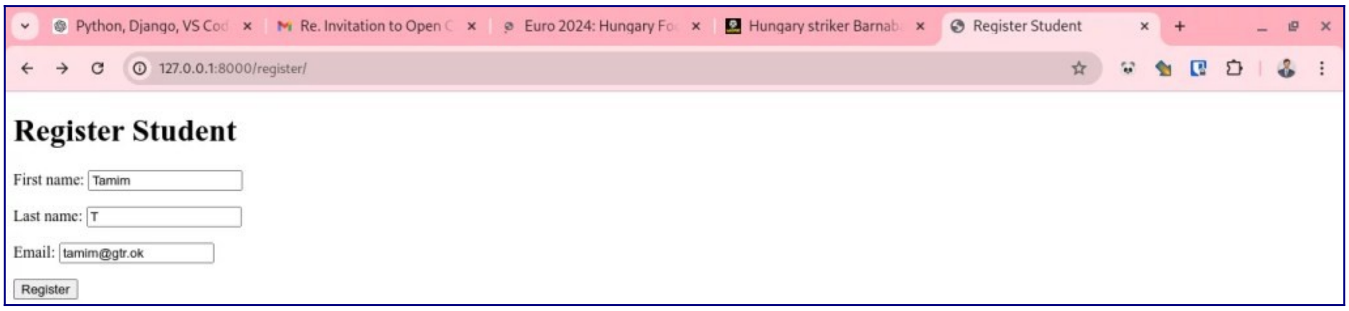
Step 9: Run the Server and Test the Application

Run the Django development server:

```
python manage.py runserver
```

Open a web browser and navigate to:

- Open `http://127.0.0.1:8000/register/` to register a student.
- After registering a student, you should be redirected to the course selection page where you can select a course for the student.
- After selecting a course, you should be redirected to the list of students enrolled in that course.



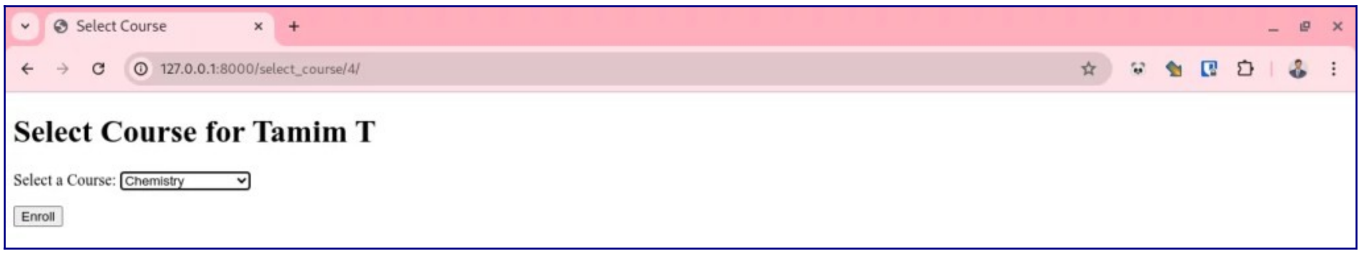
Register Student

First name: Tamim

Last name: T

Email: tamim@gtr.ok

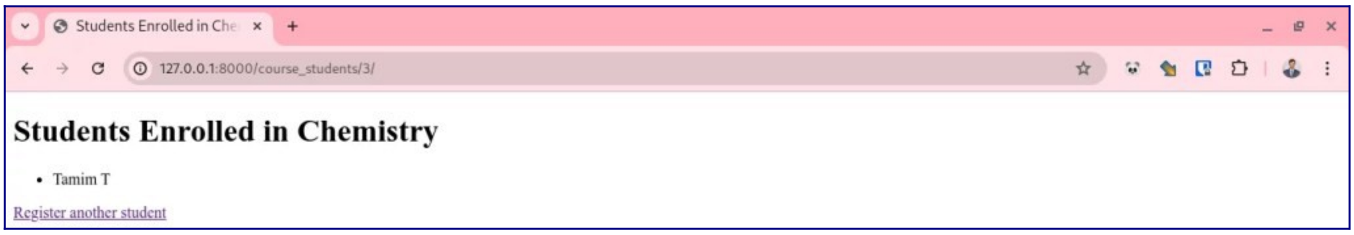
Register



Select Course for Tamim T

Select a Course: Chemistry

Enroll



Students Enrolled in Chemistry

- Tamim T

Register another student

Summary

By following these steps, you ensure that some initial courses are added to the database, allowing the course selection drop-down to display options when registering a student. This complete solution handles student registration, course selection, and displaying students registered for any selected course, ensuring that the functionality is correct and the initial data is in place.

Module 3

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

To register the `Student` and `Course` models in the Django admin interface, perform the necessary migrations, and illustrate data entry through admin forms, follow these steps:

Step 1: Register Models in Admin Interface

Edit `admin.py` in the `courses` app to register the models:

```
# courses/admin.py

from django.contrib import admin
from .models import Student, Course

class StudentAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')
    search_fields = ('first_name', 'last_name', 'email')

class CourseAdmin(admin.ModelAdmin):
    list_display = ('name', 'description')
    search_fields = ('name',)
    filter_horizontal = ('students',)

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

Step 2: Perform Migrations

Create the initial migration files for the models:

```
python manage.py makemigrations
```

Apply the migrations to update the database schema:

```
python manage.py migrate
```

Step 3: Illustrate Data Entry Through Admin Forms

Create a superuser to access the admin interface:

```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account (enter a username, email, and password).

Run the Django development server:

```
python manage.py runserver
```

Open a web browser and navigate to the admin interface:

- Go to `http://127.0.0.1:8000/admin/`
- Log in with the superuser account you just created.

Add data using the admin interface:

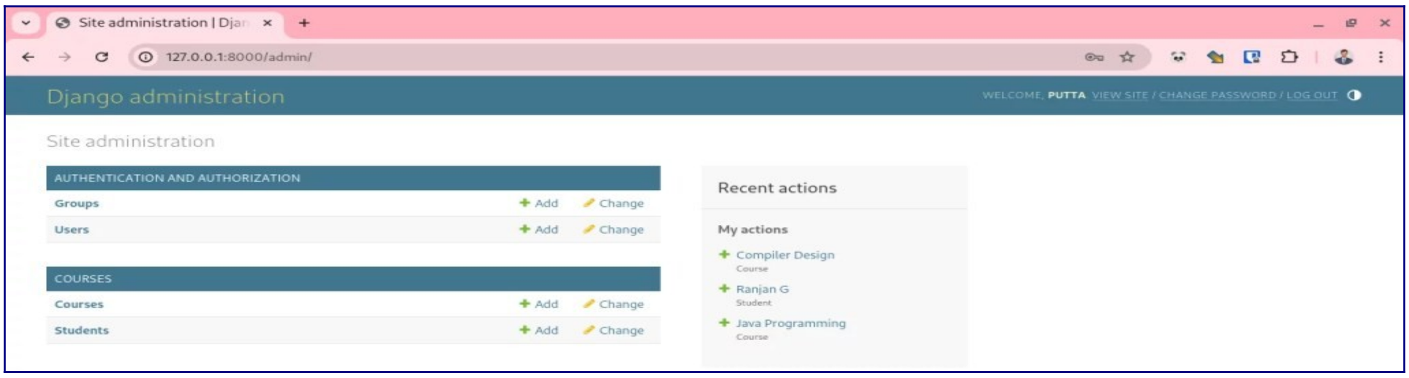
- After logging in, you will see the `Students` and `Courses` models registered in the admin interface.

Adding Students:

- Click on “Students” in the admin interface.
- Click on “Add Student” to add a new student.
- Fill in the “First name,” “Last name,” and “Email” fields.
- Click “Save” to create the student.

Adding Courses:

- Click on “Courses” in the admin interface.
- Click on “Add Course” to add a new course.
- Fill in the “Name” and “Description” fields.
- In the “Students” field, you can select students to enroll in this course.
- Click “Save” to create the course.



Site administration | Djan

127.0.0.1:8000/admin/

Django administration

WELCOME, **PUTTA** VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

COURSES		
Courses	+ Add	Change
Students	+ Add	Change

Recent actions

My actions

+ Compiler Design
  Course
+ Ranjan G
  Student
+ Java Programming
  Course

Summary

By following these steps, you’ve successfully registered the `Student` and `Course` models in the Django admin interface, performed the necessary migrations, and illustrated how to enter data through admin forms. Here is a summary of what we did:

- Registered the `Student` and `Course` models in the admin interface with customized display and search options.
- Created and applied migrations to update the database schema.
- Created a superuser to access the admin interface.
- Added data for students and courses using the Django admin interface.

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

To develop a Django project with a `Project` model containing fields for topic, languages used, and duration, and a `Student` model with a ForeignKey relationship to `Project`, we'll follow these steps:

Step 1: Set Up the Django Project and App

Create a Django project:

```
django-admin startproject student_project
cd student_project
```

Create a Django app named `registration`:

```
python manage.py startapp registration
```

Add `registration` to `INSTALLED_APPS` in `student_project/settings.py`:

```
# student_project/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'registration',  # Add your app here
]
```

Step 2: Create Models

Edit `models.py` in the `registration` app to create `Student` and `Project` models:

```
# registration/models.py

from django.db import models

class Project(models.Model):
    topic = models.CharField(max_length=200)
    languages_used = models.CharField(max_length=200)
    duration = models.CharField(max_length=100)

    def __str__(self):
        return self.topic

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    project = models.OneToOneField(Project, on_delete=models.CASCADE, null=True,
blank=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Create and apply the migrations:

```
python manage.py makemigrations
python manage.py migrate
```

Step 3: Create Forms

Create a `forms.py` file in the `registration` app to handle the registration form:

```
# registration/forms.py

from django import forms
from .models import Student, Project

class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ['topic', 'languages_used', 'duration']

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['first_name', 'last_name', 'email']
```

Step 4: Create Views

Edit `views.py` to create views for student and project registration:

```
# registration/views.py

from django.shortcuts import render, redirect
from .forms import StudentForm, ProjectForm
from .models import Student

def register_student(request):
    if request.method == 'POST':
        student_form = StudentForm(request.POST)
        project_form = ProjectForm(request.POST)
        if student_form.is_valid() and project_form.is_valid():
            project = project_form.save()
            student = student_form.save(commit=False)
            student.project = project
            student.save()
            return redirect('student_list')
    else:
        student_form = StudentForm()
        project_form = ProjectForm()
    return render(request, 'registration/register_student.html', {'student_form':
student_form, 'project_form': project_form})

def student_list(request):
    students = Student.objects.all()
    return render(request, 'registration/student_list.html', {'students': students})
```

Step 5: Configure URLs

Create a `urls.py` file in the `registration` app:

```
# registration/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register_student, name='register_student'),
    path('students/', views.student_list, name='student_list'),
]
```

Include the `registration` URLs in the main `urls.py` file:

```
# student_project/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('registration/', include('registration.urls')),  # Include the app URLs
]
```

Step 6: Create Templates

Create a directory named `templates` in the `registration` app and create subdirectories for the templates:

```
mkdir -p registration/templates/registration
```

Create `register_student.html` template:

```
<!-- registration/templates/registration/register_student.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Register Student Project</title>
</head>
<body>
    <h1>Register Student Project</h1>
    <form method="post">
        {% csrf_token %}
        {{ student_form.as_p }}
        {{ project_form.as_p }}
        <button type="submit">Register</button>
    </form>
</body>
</html>
```

Create `student_list.html` template:

```
<!-- registration/templates/registration/student_list.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Student Projects List</title>
</head>
<body>
    <h1>Registered Student Projects</h1>
    <ul>
        {% for student in students %}
            <li>{{ student.first_name }} {{ student.last_name }} - {{
student.project.topic }}</li>
            <ul>
                    <li>Languages used : {{student.project.languages_used}}</li>
                    <li>Project Duration : {{student.project.duration}}</li>
            </ul>
        {% endfor %}
    </ul>
    <a href="{% url 'register_student' %}">Register another student project</a>
</body>
</html>
```

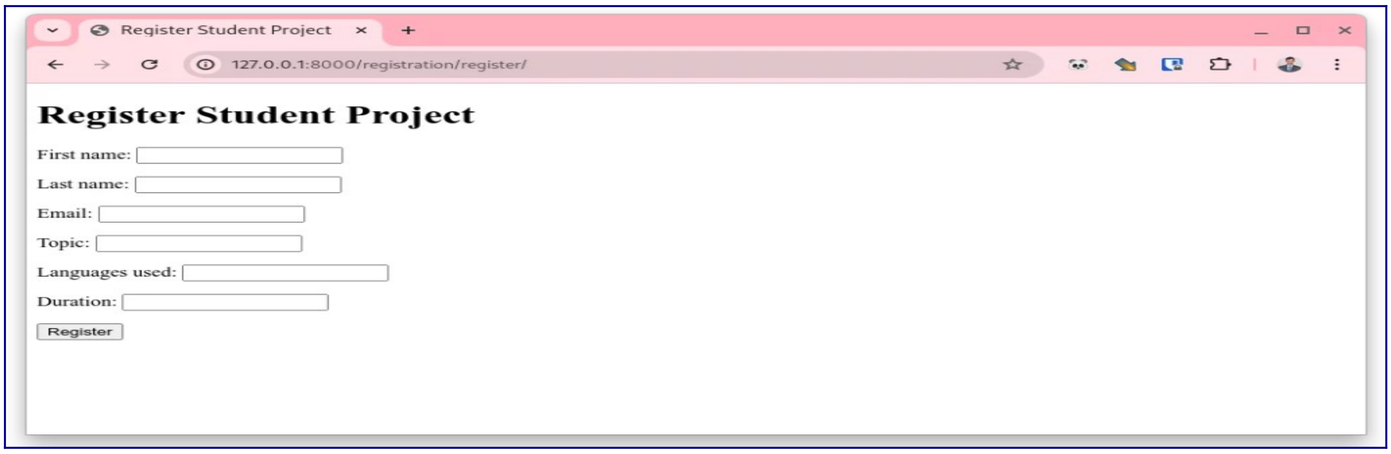
Step 7: Run the Server and Test the Application

Run the Django development server:

```
python manage.py runserver
```

Open a web browser and navigate to:

- `http://127.0.0.1:8000/registration/register/` to register a student and their project.
- `http://127.0.0.1:8000/registration/students/` to view the list of registered students and their projects.



Register Student Project

First name:

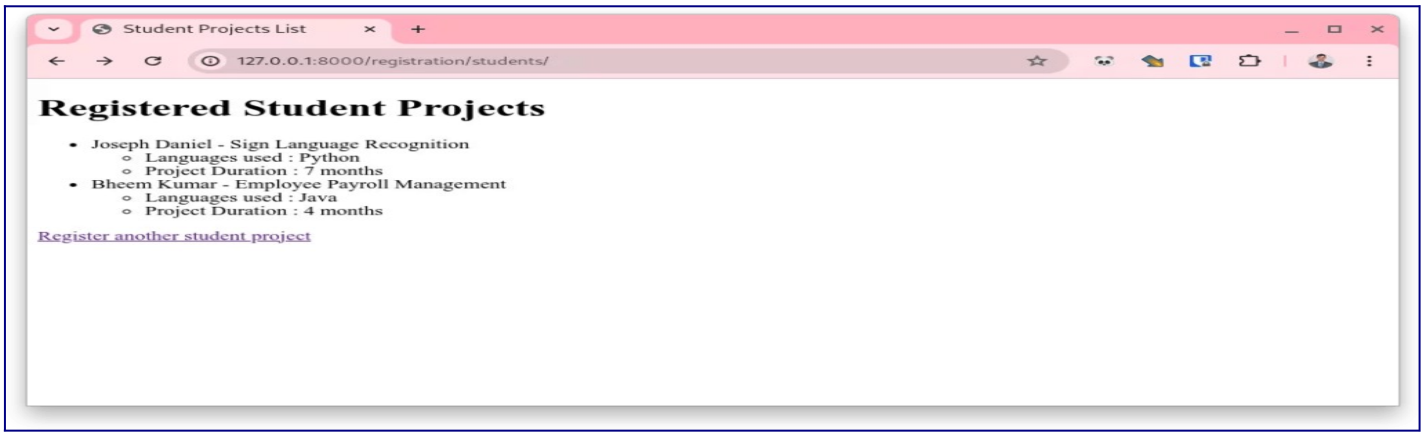
Last name:

Email:

Topic:

Languages used:

Duration:



Registered Student Projects

- Joseph Daniel - Sign Language Recognition
  - Languages used : Python
  - Project Duration : 7 months
- Bheem Kumar - Employee Payroll Management
  - Languages used : Java
  - Project Duration : 4 months

Register another student project

Summary

By following these steps, you create a Django app that allows student registration along with project details (topic, languages used, and duration). This solution includes model definitions, forms, views, templates, and URL configurations to achieve the desired functionality.