

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI**



**BANGALORE INSTITUTE OF TECHNOLOGY
K.R. ROAD, V.V PURAM, BANGALORE – 560 004**

**DEPARTMENT OF
INFORMATION SCIENCE AND ENGINEERING**



SUBJECT CODE: 21CS62

FULLSTACK DEVELOPMENT LABORATORY MANUAL

As per Choice Based Credit System Scheme (CBCS)

FOR VI SEMESTER CSE/ISE AS PRESCRIBED BY VTU

Effective from the Academic year 2023-2024

Prepared By:
Dr. Mercy S, Ph.D
Associate Professor
Dept. of ISE, BIT

BANGALORE INSTITUTE OF TECHNOLOGY

VISION

- To Establish and Develop the Institute as a Center of Higher Learning ever abreast with the Expanding horizon of knowledge in the field of Engineering and Technology, with entrepreneurial thinking, leadership excellence for life-long success and solve societal problems.

MISSION

- Provide high quality education in the Engineering disciplines from the undergraduate through doctoral levels with creative academic and professional programs.
- Develop the Institute as a leader in Science, Engineering, Technology, Management and Research and apply knowledge for the benefit of society.
- Establish mutual beneficial partnership with Industry, Alumni, Local, State and Central Governments by Public Services Assistance and Collaborative Research.
- Inculcate personality development through sports, cultural and extracurricular activities and engage in the social, economic and professional challenges.

Bangalore Institute of Technology
K.R. Road, V.V. Pura, Bengaluru 560004
Department of Information Science and Engineering

VISION:

Empower every student to be innovative, creative and productive in the field of Information Technology by imparting quality technical education, developing skills and inculcating human values.

MISSION:

1. To evolve continually as a centre of excellence in offering quality Information Technology **Education.**
2. To nurture the students to meet the global competency in industry for **Employment.**
3. To promote collaboration with industry and academia for constructive Interaction to empower **Entrepreneurship.**
4. To provide reliable, contemporary and integrated technology to support and Facilitate **Teaching, Learning, Research and Service.**

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

PEO-1	Uplift the students through Information Technology Education.
PEO-2	Provide exposure to emerging technologies and train them to Employable in multi-disciplinary industries.
PEO-3	Motivate them to become good professional Engineers and Entrepreneur.
PEO-4	Inspire them to prepare for Higher Learning and Research.

PROGRAM SPECIFIC OUTCOMES (PSOs):

1. To provide our graduates with **Core Competence** in **Information Processing and Management.**
2. To provide our graduates with **higher learning in Computing Skills.**

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological.

Bangalore Institute of Technology
K.R. Road, V.V. Pura, Bengaluru - 560004
Department of Information Science and Engineering

1. PREREQUISITES:

- Basic knowledge about computer science.
- Fundamental knowledge on front end and back end tools.
- Requires the information about web development.
- Knowledge of Python programming.
- Basic knowledge about the database and its connectivity.

2. COURSE LEARNING OBJECTIVES:

The main objectives of this course are to,

1. Explain the use of learning full stack web development.
2. Explain the core concepts of both front end and back end programming.
3. To get familiar with the latest web development technologies.
4. Illustrate models, views and templates with their connectivity in Django for full stack web development.
5. Demonstrate the use of state management and admin interfaces automation in Django.
6. Design and implement Django apps containing dynamic pages with SQL databases.

3. COURSE OUTCOMES

The students should be able to:

1. Understand the working of MVT based full stack web development with Django.
2. Apply the Django framework libraries to render nonHTML contents like CSV and PDF. Implement jQuery based AJAX integration to Django Apps to build responsive full stack web application.
3. Analyze the role of template inheritance and generic views for developing full stack web application.
4. Design a web page and implement a full Models-Views-Templates structure for an application using various tools.

4. RESOURCES REQUIRED:

- Hardware resources
 - Desktop PC
 - Windows / Linux operating system
- Software resources
 - Python
 - Visual Studio Code
 - Django

5. SUGGESTED LEARNING RESOURCES:

Textbooks

1. Adrian Holovaty, Jacob Kaplan Moss, The Definitive Guide to Django: Web Development Done Right, Second Edition, Springer-Verlag Berlin and Heidelberg GmbH & Co. KG Publishers, 2009.
2. Jonathan Hayward, Django Java Script Integration: AJAX and jQuery, First Edition, Pack Publishing, 2011

Reference Book

1. Aidas Bendroraitis, Jake Kronika, Django 3 Web Development Cookbook, Fourth Edition, Packt Publishing, 2020
2. William Vincent, Django for Beginners: Build websites with Python and Django, First Edition, Amazon Digital Services, 2018
3. Antonio Mele, Django3 by Example, 3rd Edition, Pack Publishers, 2020
4. Arun Ravindran, Django Design Patterns and Best Practices, 2nd Edition, Pack Publishers, 2020.
5. Julia Elman, Mark Lavin, Light weight Django, David A. Bell, 1st Edition, Oreily Publications, 2014

Web links and Video Lectures (e-Resources):

1. MVT architecture with Django: <https://freevideolectures.com/course/3700/django-tutorials>
2. Using Python in Django: <https://www.youtube.com/watch?v=2BqoLiMT3Ao>
3. Model Forms with Django: <https://www.youtube.com/watch?v=gMM1rtTwKxE>
4. Real time Interactions in Django: <https://www.youtube.com/watch?v=3gHmfoeZ45k>
5. AJAX with Django for beginners: <https://www.youtube.com/watch?v=3VaKNyjlxAU>

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

Real world problem solving - applying the Django framework concepts and its integration with AJAX to develop any shopping website with admin and user dashboards.

7. EVALUATION SCHEME

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures not less than 35% (18 Marks out of 50) in the semester-end examination (SEE), and a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation:

Three Unit Tests each of 20 Marks (duration 01 hour)

1. First test at the end of 5th week of the semester
2. Second test at the end of the 10th week of the semester
3. Third test at the end of the 15th week of the semester

Two assignments each of 10 Marks

4. First assignment at the end of 4th week of the semester
5. Second assignment at the end of 9th week of the semester

Practical Sessions need to be assessed by appropriate rubrics and viva-voce method. This will contribute to 20 marks.

- Rubrics for each Experiment taken average for all Lab components – 15 Marks.
- Viva-Voce– 5 Marks (more emphasized on demonstration topics)

The sum of three tests, two assignments, and practical sessions will be out of 100 marks and will be **scaled down to 50 marks**

(to have a less stressed CIE, the portion of the syllabus should not be common /repeated for any of the methods of the CIE. Each method of CIE should have a different syllabus portion of the course).

CIE methods /question paper has to be designed to attain the different levels of Bloom's taxonomy as per the outcome defined for the course.

Semester End Examination:

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the subject (duration 03 hours)

1. The question paper will be set for 100 marks. The question paper will have ten questions. Each question is set for 20 marks. Marks scored shall be proportionally reduced to 50 marks
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), should have a mix of topics under that module.

The students have to answer 5 full questions, selecting one full question from each module

FULLSTACK DEVELOPMENT SEMESTER – VI			
Course Code	21CS62	CIE Marks	50
Number of Contact Hours/Week	0:0:2	SEE Marks	50
Programs List:			
Module-1: MVC based Web Designing			
1	Installation of Python, Django and Visual Studio code editors can be demonstrated.		
2	Creation of virtual environment, Django project and App should be demonstrated		
3	Develop a Django app that displays current date and time in server		
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.		
Module-2: Django Templates and Models			
1	Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event		
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.		
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.		
Module-3: Django Admin Interfaces and Model Forms			
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.		
Module-4: Generic Views and Django State Persistence			
1	For students enrolment developed in Module 2, create a generic class view which displays list of students and detail view that displays student details for any selected student in the list.		
2	Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component.		
Module-5: jQuery and AJAX Integration in Django			
1	Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.		
2	Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.		

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

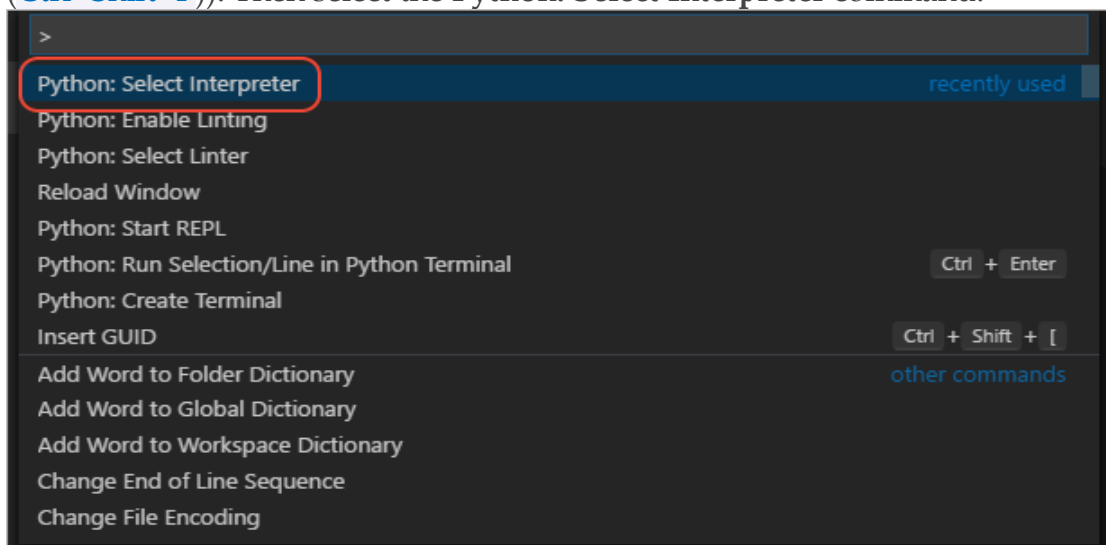
Follow these steps

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`:

Django installation

7. Create a New Terminal: In Menu Terminal -> New Terminal option and type following command:
PS C:\Users\ISE\fsd> `pip install django`

Creating project:

1. Create a django project -

Type following command in the terminal opened:

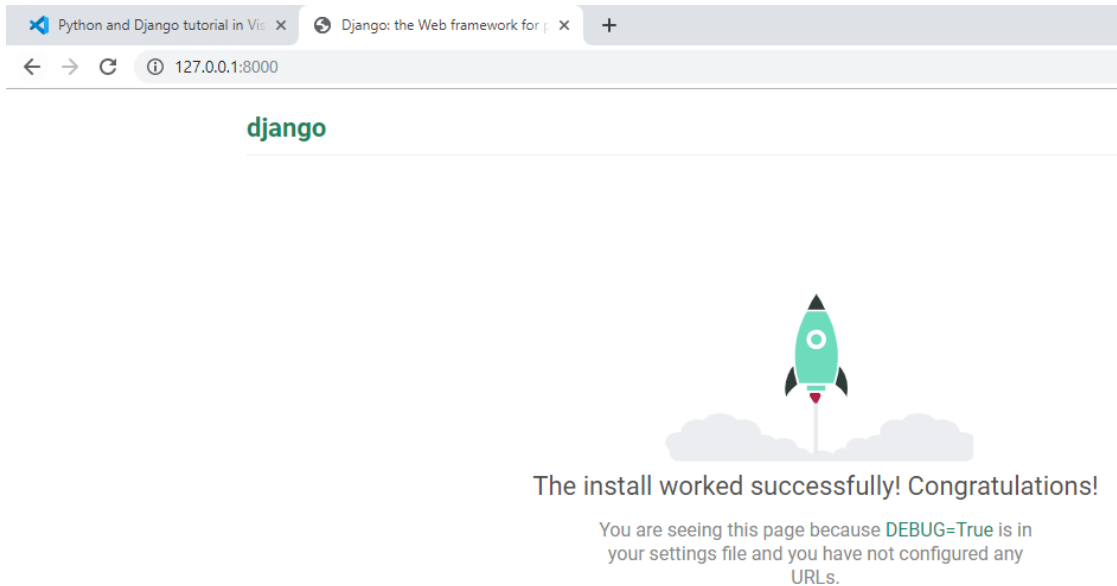
`django-admin startproject first .`

(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 `first`, 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 hello
```

5. The command creates a folder called `hello` 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

Create an app by name lab3

```
PS C:\Users\ISE\fsd> python manage.py startapp lab3
```

In lab3 subfolder, make following changes to views.py:

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

# 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)
```

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

```
from django.contrib import admin
from django.urls import path
from lab3.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 lab3 subfolder, make following changes to views.py:

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

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 4 hour(s), it will be
%s.</h1></body></html>"% (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></body></html>"% (dt,)
    return HttpResponse(html)

def four_hours_ahead_dynamic(request,offset):
    dt = datetime.datetime.now() + datetime.timedelta(hours=offset)
    html="<html><body><h1> After %s hours(s), It will be
%s.</h1></body></html>" % (offset,dt)
    return HttpResponse(html)

def four_hours_before_dynamic(request,offset):
    dt = datetime.datetime.now() + datetime.timedelta(hours=-offset)
    html="<html><body><h1> before %s hours(s), It will be
%s.</h1></body></html>" % (offset,dt)
    return HttpResponse(html)

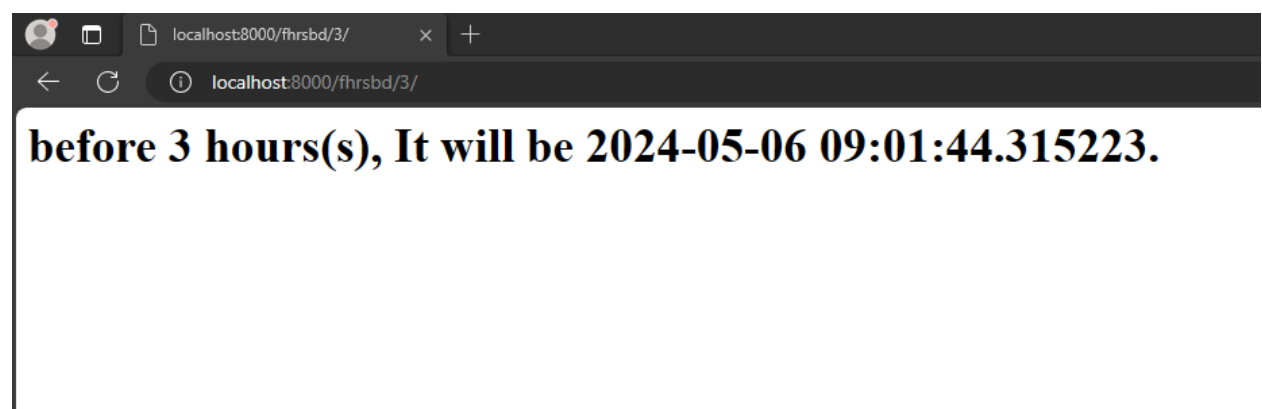
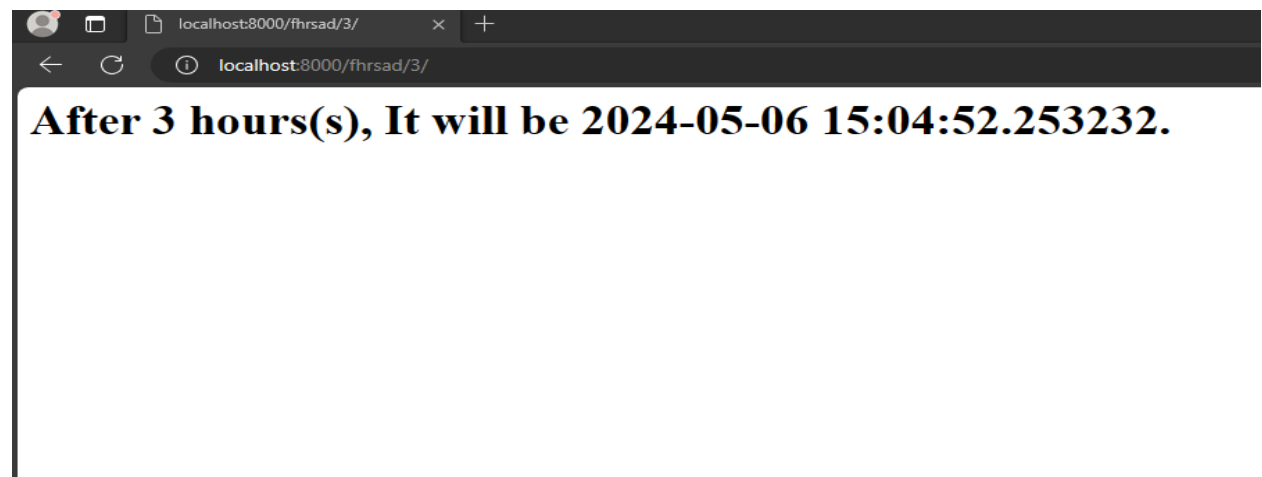
# Create your views here.
```

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

```
from django.contrib import admin
from django.urls import path
from lab3.views import current_datetime, four_hours_ahead,
four_hours_before, four_hours_ahead_dynamic, four_hours_before_dynamic

urlpatterns = [
    path('admin/', admin.site.urls),
    path('cdt/', current_datetime),
    path('fhresa/', four_hours_ahead),
    path('fhrsb/', four_hours_before),
    path('fhrsad/<int:offset>', four_hours_ahead_dynamic),
    path('fhrsbd/<int:offset>', four_hours_before_dynamic),
]
```

OUTPUT:



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

Create an app by name lab21

In lab21 subfolder, make following changes to views.py:

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import loader
```

Create your views here.

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

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

```
from django.contrib import admin
from django.urls import path
from lab21.views import show_list
```

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

Create subfolder templates inside lab21 subfolder

Make following changes to settings.py in project named first

```
import os
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,
'lab21/templates'), os.path.join(BASE_DIR),],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

In the templates subfolder, create html file: **list.html** with following HTML code:

```
<html>
  <body>
    <h1>Unordered List of Fruits</h1>
    <ul>
      <li>Apple</li>
      <li>Banana</li>
      <li>Orange</li>
      <li>Mango</li>
      <li>Pineapple</li>
    </ul>
    <h1>Ordered List of Students</h1>
    <ol>
      <li>Tinku</li>
      <li>Dinku</li>
      <li>Kinku</li>
      <li>Ninku</li>
      <li>Pinku</li>
    </ol>
  </body>
</html>
```

OUTPUT:

← → ↻ ⓘ 127.0.0.1:8000/showlist/

Unordered List of Fruits

- Apple
- Banana
- Orange
- Mango
- Pineapple

Ordered List of Students

1. Tinku
2. Dinku
3. Kinku
4. Ninku
5. Pinku

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.

Store following html files inside templates subfolder:

layout.html code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>{% block title %}{% endblock %}</title>

</head>

<body>
<div class="navbar">
  <a href="/home" class="navbar-brand">Home</a>
  <a href="/about" class="navbar-item">About</a>
  <a href="/contact" class="navbar-item">Contact</a>
</div>

<div class="body-content">
  <br><br>
  {% block content %}
  {% endblock %}
  <!-- <hr/> -->
  <footer>
    <p>&copy; Dept. of ISE, BIT, 2024</p>
  </footer>
</div>
</body>
</html>
```

home.html

```
{% extends "layout.html" %}
{% block title %}
Home
{% endblock %}

{% block content %}
This is the home page of the website
{% endblock %}
```

aboutus.html

```
{% extends "layout.html" %}
{% block title %}
ABOUT US

{% endblock %}

{% block content %}
We are the developers of this page

{% endblock %}
```

contactus.html

```
{% extends "layout.html" %}
{% block title %}
CONTACT US

{% endblock %}

{% block content %}
Contact Information is given here

{% endblock %}
```

Changes to **views.py** inside lab21 subfolder:

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import loader

# Create your views here.
def aboutus(request):
    return render(request, 'aboutus.html')

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

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

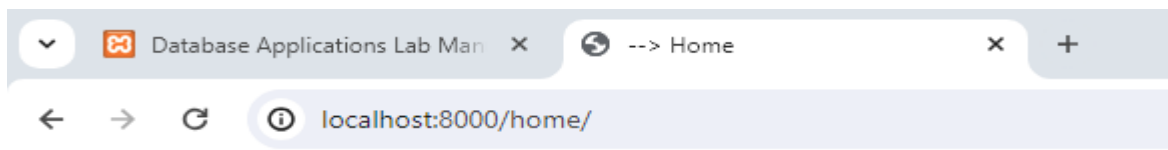
changes to `urls.py` in project first:

```
from django.contrib import admin
from django.urls import path
from lab21.views import show_list, aboutus, home, contactus

urlpatterns = [

    path('home/',home),
    path('about/',aboutus),
    path('contact/',contactus),
]
```

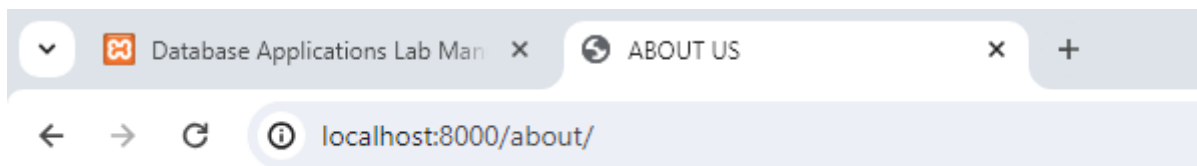
OUTPUT:



[Home](#) [About](#) [Contact](#)

This is the home page of the website

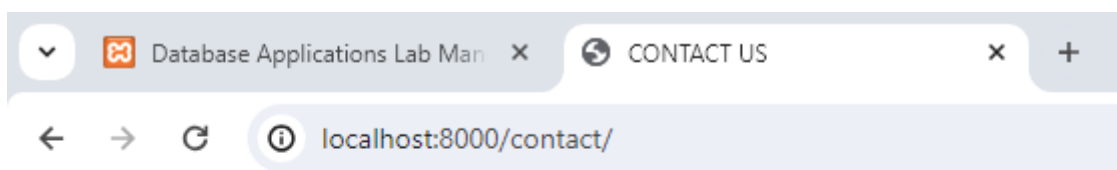
© Dept. of ISE, BIT, 2024



[Home](#) [About](#) [Contact](#)

We are the developers of this page

© Dept. of ISE, BIT, 2024



[Home](#) [About](#) [Contact](#)

Contact Information is given here

© Dept. of ISE, BIT, 2024

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.

Create another app lab22

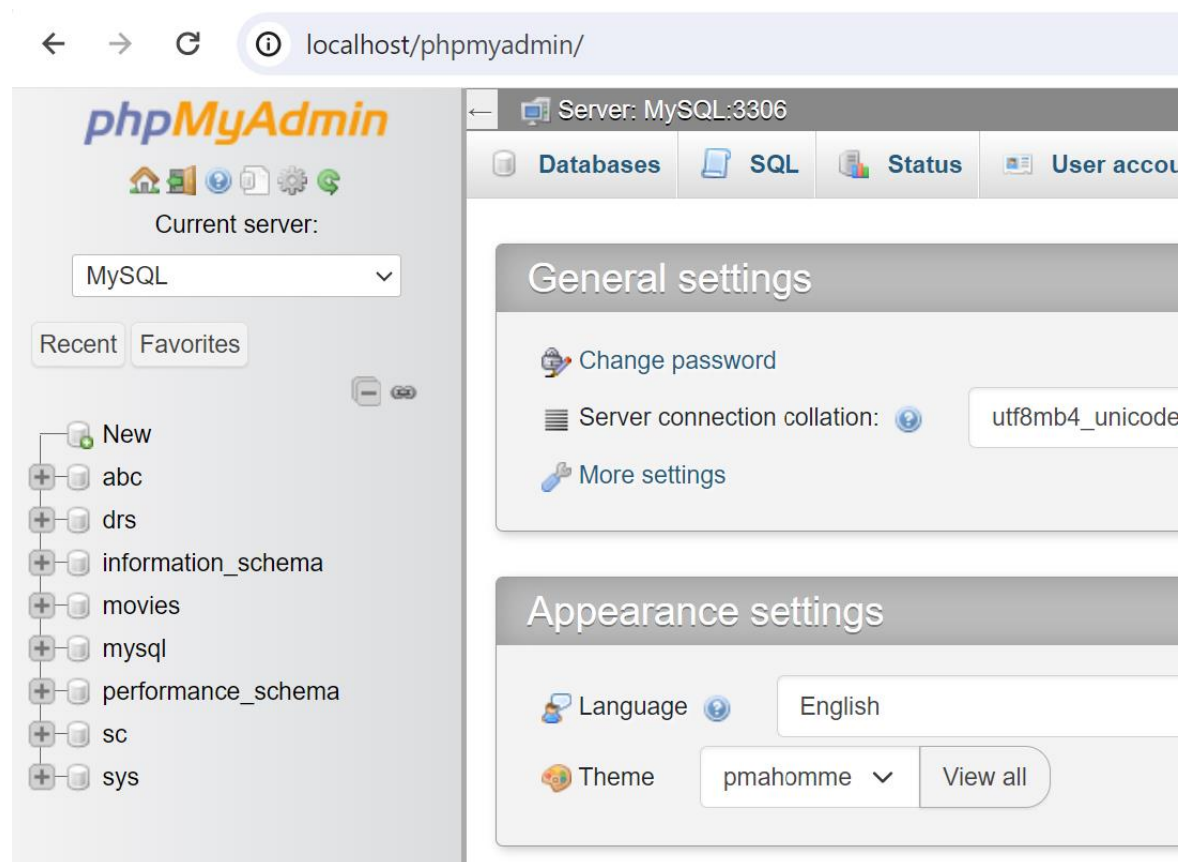
Prepare the backend (database) for this app

Install MYSQL using WAMP

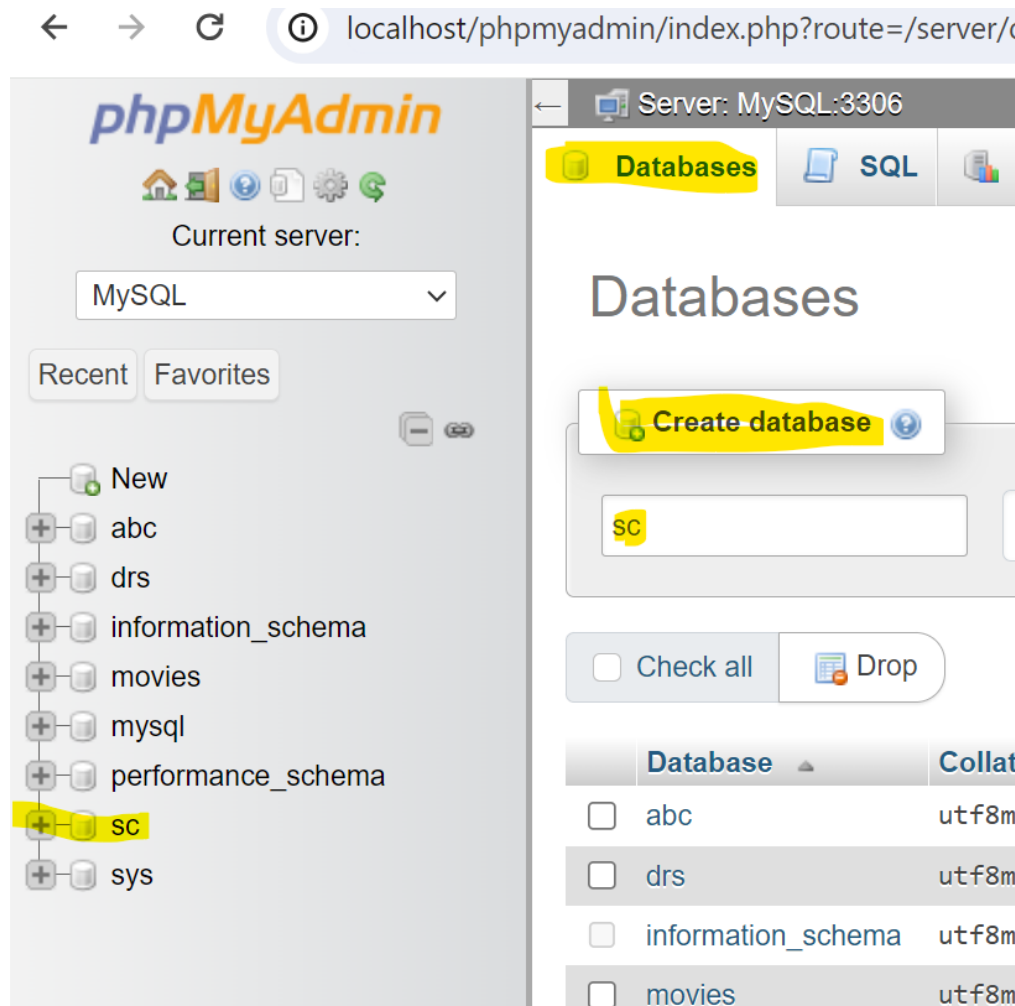
One can download WAMP at: <https://sourceforge.net/projects/wampserver/>

After installation of WAMP, open phpMyAdmin:

Default username is: **root**
No Password



Create a database using Databases tab. Let the name of database be sc



Open a new terminal and type the following:

pip install mysqlclient

PS C:\Users\ISE\fsd> pip install mysqlclient

Make following changes in settings.py of first folder:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'sc',
        'HOST': 'localhost',
        'USER': 'root',
        'PASSWORD': '',
        'PORT': 3306,
    }
}
```

Then, in lab22 subfolder, make following changes to models.py:

```
from django.db import models

# Create your models here.
class Course(models.Model):
    cname = models.CharField(max_length=10)
    ccode = models.CharField(max_length=30)
    credits = models.IntegerField()
    def __str__(self):
        return self.cname
class Students(models.Model):
    name = models.CharField(max_length=30)
    usn = models.CharField(max_length=50)
    sem = models.IntegerField()
    branch = models.CharField(max_length=30)
    sce = models.ManyToManyField(Course)
    def __str__(self):
        return self.name
```

In your settings.py file, locate the INSTALLED_APPS list and add 'lab22' to it like this:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'lab22',
]
```

Then open a new terminal and type following commands one by one

```
python manage.py makemigrations lab22
python manage.py sqlmigrate lab22 0001
python manage.py migrate
```

After these commands, all backend tables will be automatically created

Using phpMyAdmin enter data for student and courses table

Student:

```
INSERT INTO `lab22_students`(`id`,`name`,`usn`,`sem`,`branch`) VALUES
(1,'Rona','R001',5,'CSE')
```

```
INSERT INTO `lab22_students`(`id`,`name`,`usn`,`sem`,`branch`) VALUES
(2,'Mona','R002',5,'CSE')
```

```
INSERT INTO `lab22_students`(`id`,`name`,`usn`,`sem`,`branch`) VALUES
(3,'Sona','R003',5,'CSE')
```

```
INSERT INTO `lab22_students`(`id`,`name`,`usn`,`sem`,`branch`) VALUES
(4,'Moni','R005',5,'ISE')
```

The screenshot shows the phpMyAdmin interface. The left sidebar displays the database structure, with the 'sc' database selected. The main panel shows the 'lab22_students' table. The table structure is defined as:

id	name	usn	sem	branch
1	Rona	R001	5	CSE
2	Mona	R002	5	CSE
3	Sona	R003	5	CSE
4	MONI	R005	5	ISE

Course:

```
INSERT INTO `lab22_course`(`id`,`cname`,`ccode`,`credits`) VALUES
(1,'DBMS','C001',4)
```

```
INSERT INTO `lab22_course`(`id`,`cname`,`ccode`,`credits`) VALUES (2,'OS','C002',3)
```

```
INSERT INTO `lab22_course`(`id`,`cname`,`ccode`,`credits`) VALUES
(3,'PYTHON','C003',4)
```

```
INSERT INTO `lab22_course`(`id`,`cname`,`ccode`,`credits`) VALUES
(4,'RUBY','C004',3)
```

The screenshot shows the phpMyAdmin interface. The left sidebar shows the database structure with 'sc' selected. The main panel shows the 'lab22_course' table with the following data:

id	cname	ccode	credits
1	DBMS	C001	4
2	OS	C002	3
3	PYTHON	C003	4
4	RUBY	C004	3

Next, inside lab22 subfolder create templates subfolder
 Inside templates subfolder, store following HTML files:

sentry.html

```
<html>
  <body>
    <form method="post" action="">
      {% csrf_token %}
      <input type="text" name="sname" placeholder="Student Name">
      <input type="text" name="cname" placeholder="Course Name">
      <input type="submit" value="Submit">

    </form>
  </body>
</html>
```

register.html

```
<html>
  <body>
    {{ message }}
  </body>
</html>
```


view.html

```

<html>
  <body>
    <h1>Select the course</h1>
    <form method="post" action="displaystudents/">
      {% csrf_token %}
      <select name="course" id="course">
        {% for course in courses %}

          <option value="{{ course.id }}">{{ course.cname }}</option>

        {% endfor %}
      </select>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>

```

displaystudents.html

```

<html>
  <body>
    <h1>List of Student Names </h1>
    <ul>
      {% for name in l %}
        <li>{{ name }}</li>
      {% endfor %}
    </ul>
  </body>
</html>

```

Make following changes in **views.py** inside lab22 subfolder:

```

from django.shortcuts import render
from lab22.models import Students, Course
from django.http import HttpResponse

# Create your views here.

def register(request):
    if request.method == 'POST':
        sname = request.POST.get('sname')
        cname = request.POST.get('cname')

```

```

students = Students.objects.filter(name=sname).values()
if students:
    print(students.first())
    sid=students.first()['id']
    s=Students.objects.get(id=sid)
    courses = Course.objects.filter(cname=cname).values()
    if courses:
        cid=courses.first()['id']
        c=Course.objects.get(id=cid)
        s.sce.add(c)
        return render(request,'register.html', {'message':
'Successfully registered'})
    else:
        return render(request,'register.html',{'message':
'Course not found'})
    else:
        return render(request,'register.html',{'message': 'Student
not found'})
    else:
        return render(request,'sentry.html')

def viewstudent(request):
    courses = Course.objects.all().values()
    if courses:
        return render(request,'view.html', {'courses':courses})
    else:
        return 'Student not found'

def displaystudents(request):
    cid = request.POST.get('course')
    students=Students.objects.all()
    l=list()

    for s in students:
        ss=s.sce.filter(id=cid).values()
        if ss:
            l.append(s.name)

    if len(l)>=1:
        return render(request,'displaystudents.html', {'l':l})
    else:
        return HttpResponse("NO Students found for the
course")

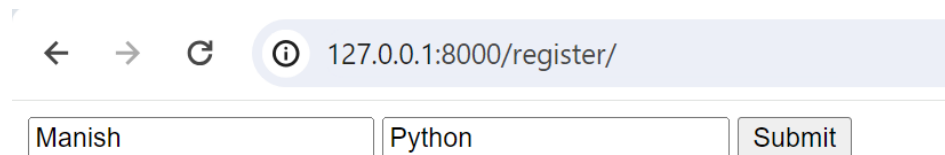
```

Make following changes to `urls.py` in first subfolder

```
from django.contrib import admin
from django.urls import path
from lab22.views import register, viewstudent, displaystudents
urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', register),
    path('viewstudent/', viewstudent),
    path('viewstudent/displaystudents/', displaystudents),
]
```

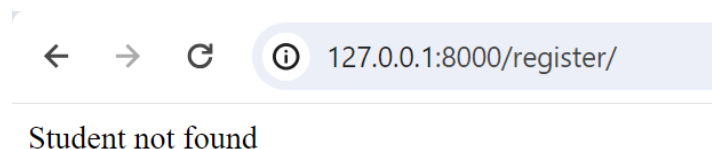
OUTPUT:

Case(i)



← → ↻ ⓘ 127.0.0.1:8000/register/

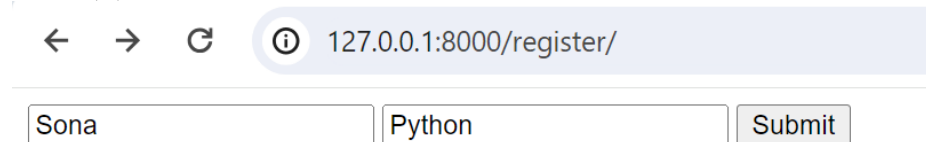
After submit is clicked:



← → ↻ ⓘ 127.0.0.1:8000/register/

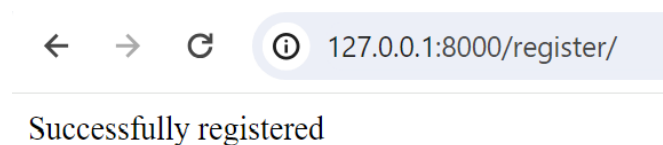
Student not found

Case (ii):



← → ↻ ⓘ 127.0.0.1:8000/register/

After submit is clicked:



← → ↻ ⓘ 127.0.0.1:8000/register/

Successfully registered

Case (iii):

← → ↻ ⓘ 127.0.0.1:8000/viewstudent/

Select the course

DBMS ▼ Submit

After clicking submit:

← → ↻ ⓘ 127.0.0.1:8000/viewstudent/displaystudents/

List of Student Names

- Rona
- Sona

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.

In the same subfolder lab22, make following changes to admin.py

```
from django.contrib import admin
from lab22.models import Students, Course

# Register your models here.
admin.site.register(Students)
admin.site.register(Course)
```

Create an admin account by typing the command

python manage.py createsuperuser

PS C:\Users\ISE\fsd> python manage.py createsuperuser

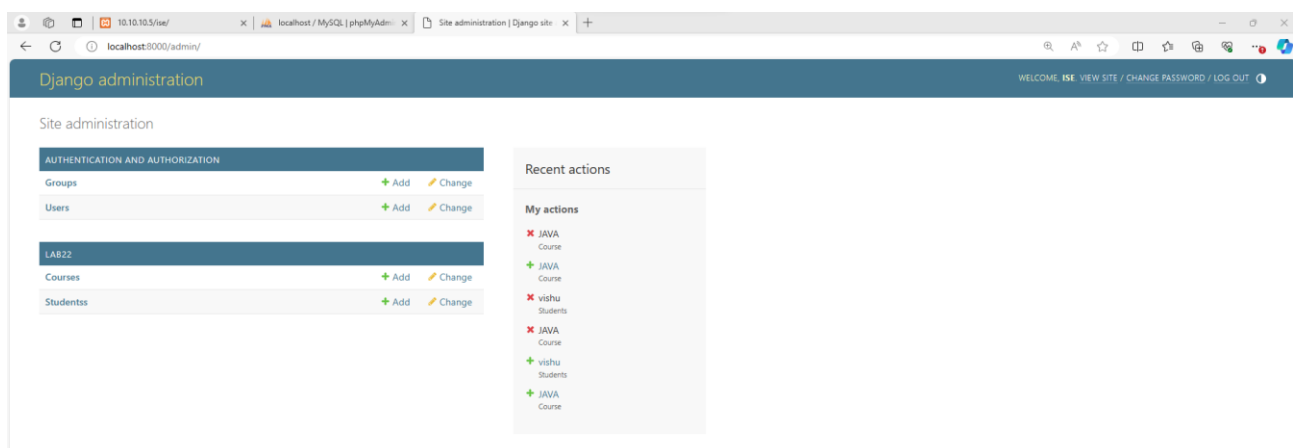
Make following changes to url.py in first folder:

```
from django.contrib import admin
urlpatterns = [

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

]
```

Then login to admin as follows:



Go to Courses and perform Insert, delete, update and search operations through form created:

The screenshot shows the Django administration interface for the 'lab22' app. The left sidebar contains a navigation menu with 'AUTHENTICATION AND AUTHORIZATION' (Groups, Users) and 'LAB22' (Courses, Students). The main content area is titled 'Select course to change' and features a search bar, an 'ADD COURSE' button, and a list of courses: COURSE, RUBY, PYTHON, OS, and DBMS. The 'Action' dropdown is set to '-----' and the 'Go' button is visible. The status '0 of 4 selected' is shown.

The screenshot shows the Django administration interface for the 'lab22' app, specifically the 'Add course' form. The left sidebar is the same as the previous screenshot. The main content area is titled 'Add course' and contains form fields for 'Cname' (filled with 'JAVA'), 'Code' (filled with 'C005'), and 'Credits' (filled with '4'). Below the form fields are three buttons: 'SAVE', 'Save and add another', and 'Save and continue editing'.

The screenshot shows the Django administration interface for the 'lab22' app, specifically the 'Select course to change' form. A green success message at the top states 'The course "JAVA" was added successfully.' The left sidebar is the same as the previous screenshots. The main content area is titled 'Select course to change' and features a search bar, an 'ADD COURSE' button, and a list of courses: COURSE, JAVA, RUBY, PYTHON, OS, and DBMS. The 'Action' dropdown is set to '-----' and the 'Go' button is visible. The status '0 of 5 selected' is shown.

10.10.10.5/ise/ localhost / MySQL | phpMyAdmin x Add course | Django site admin x

localhost:8000/admin/lab22/course/add/

Django administration

Home > Lab22 > Courses > Add course

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

LAB22

Courses + Add

Studentss + Add

Add course

Cname:

Ccode:

Credits:

SAVE Save and add another Save and continue editing

10.10.10.5/ localhost / MySQL | phpMyAdmin x Select course to change | Django x

localhost:8000/admin/lab22/course/

Django administration

Home > Lab22 > Courses

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

LAB22

Courses + Add

Studentss + Add

✓ The course "JAVA" was added successfully.

Select course to change

Action: Go 0 of 5 selected

☐ COURSE

☐ JAVA

☐ RUBY

☐ PYTHON

☐ OS

☐ DBMS

5 courses

127.0.0.1:8000/admin/lab22/course/

Django administration

Home > Lab22 > Courses

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

LAB22

Courses + Add

Studentss + Add

Select course to change

ADD COURSE +

Action: Go 0 of 4 selected

☐ COURSE

☐ RUBY

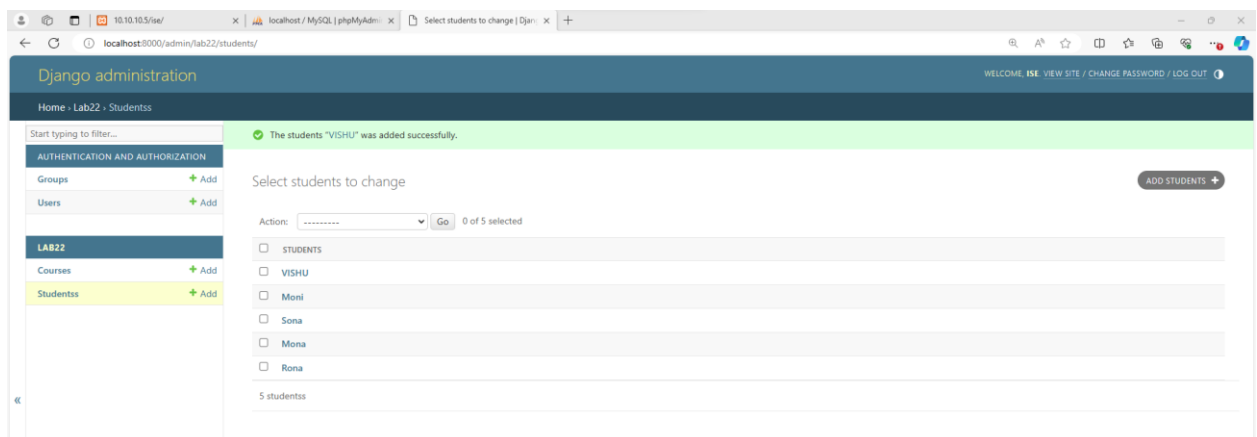
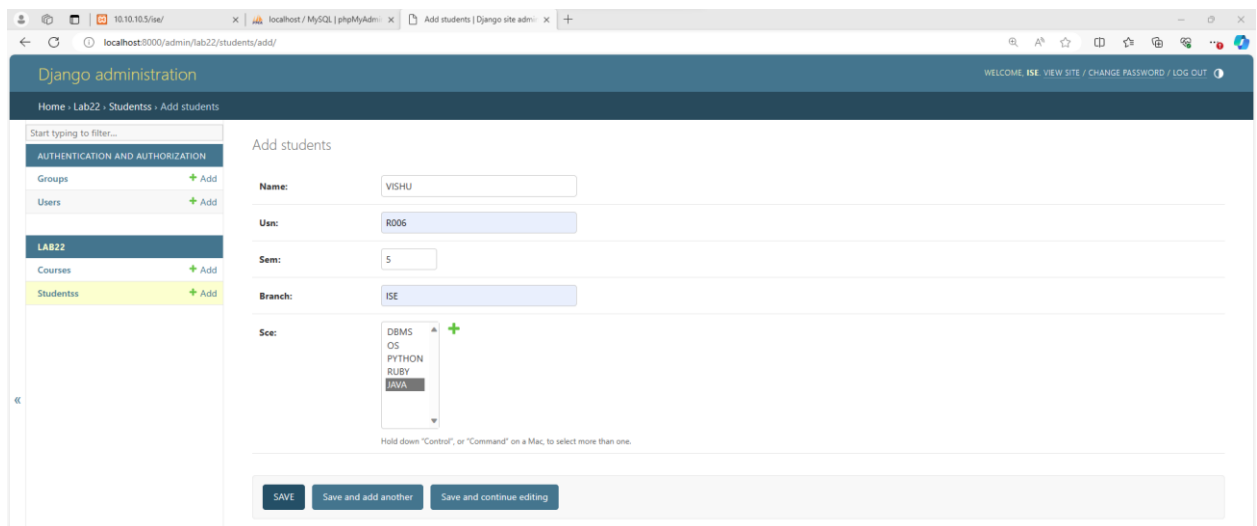
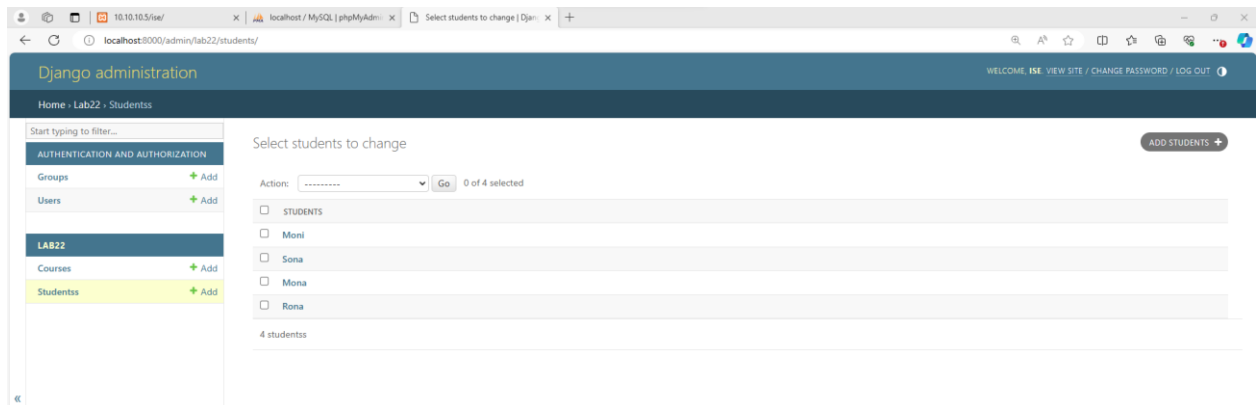
☐ PYTHON

☐ OS

☐ DBMS

4 courses

Similarly explore Students and perform insert, delete, update and search operations



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

Make following changes to models.py in lab22 subfolder:

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

Create your models here.

```
class Project(models.Model):
    student=models.ForeignKey(Students,on_delete=models.CASCADE)
    topic=models.CharField(max_length=100)
    languages=models.CharField(max_length=100)
    duration=models.IntegerField()
    def __str__(self):
        return self.topic

class ProjectReg(ModelForm):
    required_css_class = 'required'
    class Meta:
        model=Project
        fields=('student','topic','languages','duration')
```

Then open a new terminal and type following commands one by one:

```
python manage.py makemigrations lab22
python manage.py sqlmigrate lab22 0001
python manage.py migrate
```

After these commands, all backend tables will be automatically created

Create a HTML file project_reg.html inside templates subfolder:

```
{% if submitted %}
    <p class="success">
        Project Registration is successful. Thank you.
    </p>
{% else %}
<form action="" method="post" novalidate>
    <table>
        {{ form.as_table }}
        <tr>
            <td>&nbsp;</td>
            <td><input type="submit" value="Submit"></td>
        </tr>
    </table>
    {% csrf_token %}
</form>
{% endif %}
```

Make following changes to views.py:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect, FileResponse
from lab22.models import ProjectReg
```

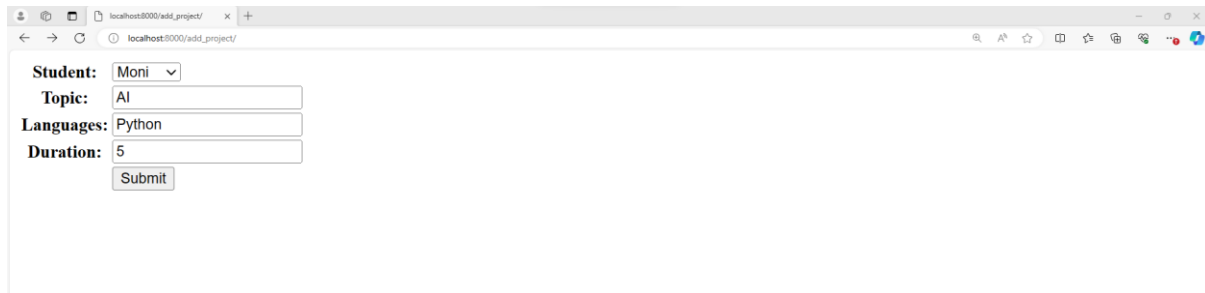
```
def add_project(request):
    submitted = False
    if request.method == 'POST':
        form = ProjectReg(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect('/add_project/?submitted=True')
    else:
        form = ProjectReg()
        if 'submitted' in request.GET:
            submitted = True
    return render(request, 'project_reg.html', {'form': form, 'submitted':
submitted})
```

Make following changes to urls.py in first folder

```
from django.contrib import admin
from django.urls import path
from lab22.views import add_project

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

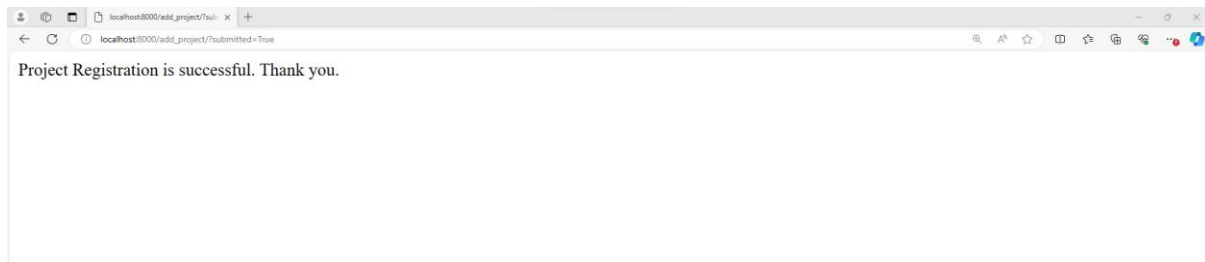
OUTPUT:



A screenshot of a web browser window displaying a project registration form. The browser's address bar shows 'localhost:8000/add_project/'. The form contains the following fields and controls:

- Student:** A dropdown menu with 'Moni' selected.
- Topic:** A text input field containing 'AI'.
- Languages:** A text input field containing 'Python'.
- Duration:** A text input field containing '5'.
- Submit:** A button labeled 'Submit'.

On adding details and clicking submit:



Module-4: Generic Views and Django State Persistence

Laboratory Component:

1. For students enrolment developed in Module 2, create a generic class view which displays list of students and detail view that displays student details for any selected student in the list.

Make following changes to views.py inside lab22

```
from django.shortcuts import render
from lab22.models import Students, Course
from django.http import HttpResponse
from django.http import HttpResponseRedirect, FileResponse
from lab22.models import ProjectReg
from django.views import generic
```

```
class StudentListView(generic.ListView):
    model = Students
    template_name = 'student_list.html'

class StudentDetailView(generic.DetailView):
    model = Students
    template_name = "student_detail.html"
```

Adding following html files to templates folder inside lab22

student_list.html

```
<h1>List of Students</h1>
{% if students_list %}
<table border>
    {% for student in students_list %}
        <tr>
            <td>
                <a href="student_detail/{{student.pk}}" target=_blank>
                    {{student.name}} </a>
            </td>
            <td> Courses:
                {% for course in student.sce.all %}
                    <span>{{course.cname}}</span>
                {% endfor %}
            </td>
        </tr>
    {% endfor %}
</table>
{% else %}
    <p>There are no students.</p>
{% endif %}
```

student_detail.html

```
<h1>Name: {{ students.name }}</h1>
<h1>Sem : {{ students.sem}}</h1>
<h1>USN : {{ students.usn }}</h1>
<h1>Branch : {{ students.branch }}</h1>
```

Make following changes to urls.py inside first folder

```
from django.urls import path
from lab22.views import StudentListView, StudentDetailView

urlpatterns = [

    path('student_list/', StudentListView.as_view()),
    path('student_list/student_detail/<int:pk>', StudentDetailView.as_view()),

]
```

OUTPUT:

← → ↻ ⓘ 127.0.0.1:8000/student_list/

List of Students

Rona	Courses: DBMS RUBY
Mona	Courses:
Sona	Courses: DBMS PYTHON
MONI	Courses: RUBY

On clicking any student name:

← → ↻ ⓘ 127.0.0.1:8000/student_list/student_detail/1/

Name: Rona

Sem : 5

USN : R001

Branch : CSE

2. Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component

Install following in a terminal before proceeding:

```
pip install reportlab
```

Make following changes to views.py inside lab22 subfolder:

```
from django.shortcuts import render
from lab22.models import Students, Course
from django.http import HttpResponseRedirect, FileResponse
from reportlab.pdfgen import canvas
```

```
import csv
def download_csv(queryset):
    opts = queryset.model._meta
    model = queryset.model
    response = HttpResponseRedirect(content_type='text/csv')
    # force download.
    response['Content-Disposition'] = 'attachment;filename=export.csv'
    # the csv writer
    writer = csv.writer(response)
    field_names = [field.name for field in opts.fields]
    # Write a first row with header information
    writer.writerow(field_names)
    for obj in queryset:
        writer.writerow([getattr(obj, field) for field in field_names])
    return response

def download(request):
    data = download_csv(Course.objects.all())

    return HttpResponseRedirect (data, content_type='text/csv')

def generate_pdf_file():
    from io import BytesIO

    buffer = BytesIO()
    p = canvas.Canvas(buffer)

    # Create a PDF document

    courses = Course.objects.all()
    p.drawString(100, 750, "Course Details")

    y = 700
```

```
for course in courses:
    p.drawString(100, y, f"Title: {course.cname}")
    p.drawString(100, y - 20, f"Code: {course.ccode}")
    p.drawString(100, y - 40, f"Credits: {course.credits}")
    y -= 60

p.showPage()
p.save()

buffer.seek(0)
return buffer

def generate_pdf(request):
    response = FileResponse(generate_pdf_file(),
                           as_attachment=True,
                           filename='course_details.pdf')

    return response
```

make following changes to urls.py inside first folder

```
from django.urls import path
from lab22.views import StudentListView, StudentDetailView, download,
generate_pdf
```

```
urlpatterns = [
    path('download/', download),
    path('generate_pdf_file/', generate_pdf),
]
```

Output:

Run URLs in browser:

<http://127.0.0.1:8000/download/>

CSV file gets downloaded

http://127.0.0.1:8000/generate_pdf_file/

PDF file gets downloaded

Module-5: jQuery and AJAX Integration in Django

Laboratory Component:

1. Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.

Create another app lab31

Open a new terminal and type following command

```
pip install --upgrade django-ajax-selects
```

Create a templates folder inside lab31

Make following changes to settings.py in first folder

```
import os
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,
'lab21/templates'),os.path.join(BASE_DIR, 'lab31/templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

Inside templates folder, create following HTML file:

sentryAJ.html

```
<html>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js">
</script>
  <body>
    <form method="post" action="">
      {% csrf_token %}
```



```

        <input type="text" name="sname" id="sname"
placeholder="Student Name"><br>
        <input type="text" name="cname" id="cname"
placeholder="Course Name"><br>
        <input type="button" value="Submit" id="submitBtn">

    </form>
    <script>

        $(document).ready(function() {

            $("#submitBtn").click(function(){
                var sname = $('#sname').val();
                var cname = $('#cname').val();

                $.ajax({
                    url: 'http://127.0.0.1:8000/registerAJ/',
                    type: 'POST',
                    data: {sname: sname, cname:cname,
csrfmiddlewaretoken: "{{ csrf_token }}",},
                    success: function(data){
                        alert(data);
                        //window.location.href = data;

                    }
                });
            });
        });

    </script>
</body>
</html>

```

Make following changes to views.py inside lab31

```

from django.shortcuts import render
from lab22.models import Students, Course
from django.http import HttpResponse

# Create your views here.
def registerAJ(request):
    if request.method == 'POST':

```

```

sname = request.POST.get('sname')
cname = request.POST.get('cname')
students = Students.objects.filter(name=sname).values()
if students:
    print(students.first())
    sid=students.first()['id']
    s=Students.objects.get(id=sid)
    courses = Course.objects.filter(cname=cname).values()
    if courses:
        cid=courses.first()['id']
        c=Course.objects.get(id=cid)
        s.sce.add(c)
        return render(request,'registerAJ.html', {'message':
'Successfully registered'})
    else:
        return render(request,'registerAJ.html',{'message':
'Course not found'})
    else:
        return render(request,'registerAJ.html',{'message':
'Student not found'})
    else:
        return render(request,'sentryAJ.html')

```

Make following changes to urls.py inside first folder

```

from django.contrib import admin
from django.urls import path
from lab22.views import register, viewstudent, displaystudents
from lab22.views import add_project
from lab22.views import StudentListView, StudentDetailView, download,
generate_pdf
from lab31.views import registerAJ, searchAJ

urlpatterns = [

    path('registerAJ/',registerAJ),

]

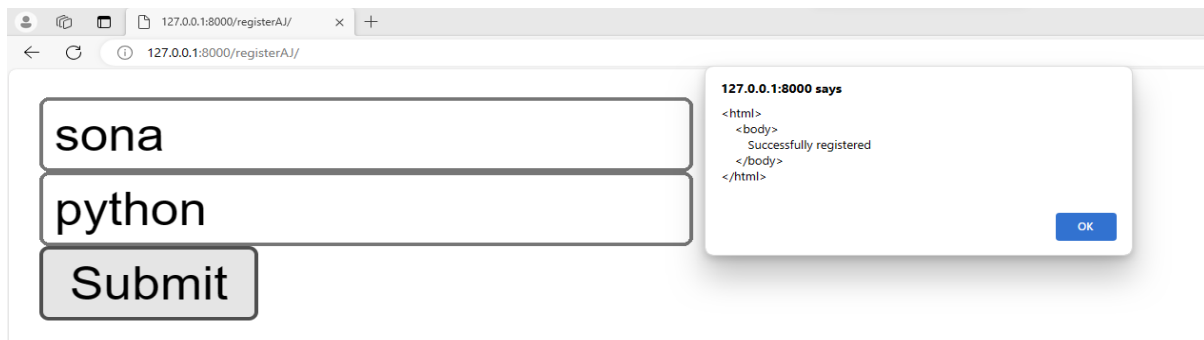
```

OUTPUT:

On clicking submit button:



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/registerAJ/`. The page contains a registration form with three input fields stacked vertically. The first field contains the text "Mona", the second field contains "Python", and the third field is a "Submit" button.



A screenshot of the same web browser window after the "Submit" button has been clicked. The form fields now contain "sona" and "python". A toast notification box is displayed on the right side of the screen, showing the message "127.0.0.1:8000 says" followed by the HTML code: `<html><body>Successfully registered</body></html>`. An "OK" button is visible at the bottom right of the notification box.

2. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

Create an HTML file search.html inside templates subfolder

```
<html>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js">
</script>
  <body>
    <input type="text" placeholder="Student Name" id="sname"><br>
    <input type="button" id="stBtn" value="Search">
    <div id="res"></div>
    <script>

      $(document).ready(function() {

        $("#stBtn").click(function(){
          var sname = $('#sname').val();

          $.ajax({
            url: 'http://127.0.0.1:8000/searchAJ/',
            type: 'POST',
            data: {sname: sname, csrfmiddlewaretoken: "{{
csrf_token }}"},
            success: function(data){
              $("#res").html(data);
              //window.location.href = data;

            }
          });
        });

      });
    </script>
  </body>
</html>
```

Make following changes to views.py inside lab31

```

from django.shortcuts import render
from lab22.models import Students,Course
from django.http import HttpResponse

def searchAJ(request):
    if request.method == 'POST':
        sname = request.POST.get('sname')

        students=Students.objects.filter(name=sname)
        l=list()
        sstr=''
        if not students.exists():
            return HttpResponse("Student not found")
        for s in students:
            ss=s.sce.all()
            print(ss)
            if ss:
                i=0
                sstr+="




```

Make following changes to urls.py inside first folder

```

from django.contrib import admin
from django.urls import path
from lab22.views import register, viewstudent, displaystudents
from lab22.views import add_project
from lab22.views import StudentListView, StudentDetailView, download,
generate_pdf
from lab31.views import registerAJ, searchAJ

urlpatterns = [

    path('registerAJ/',registerAJ),
    path('searchAJ/',searchAJ),

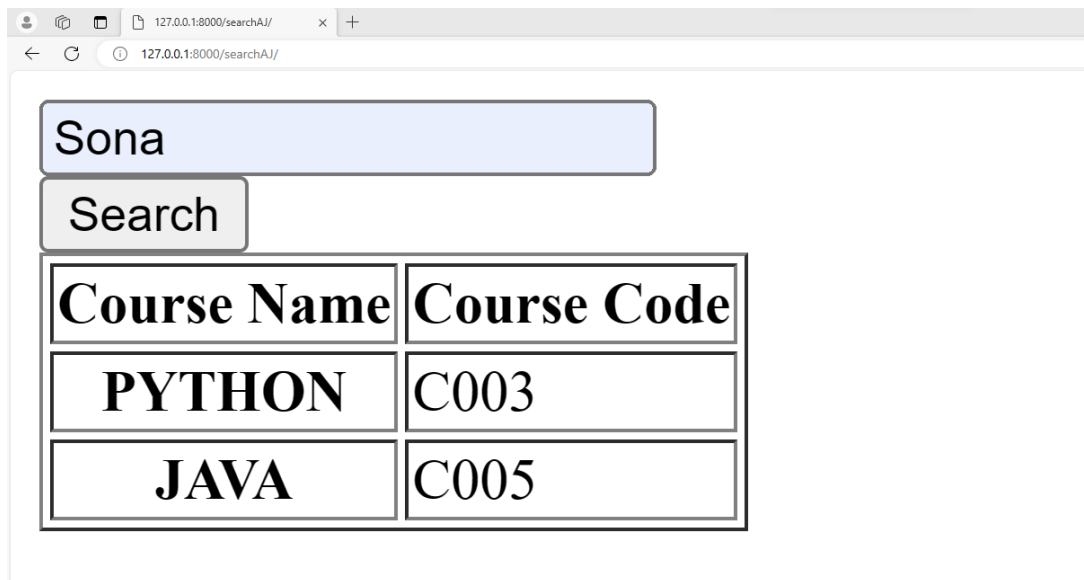
]

```

OUTPUT:

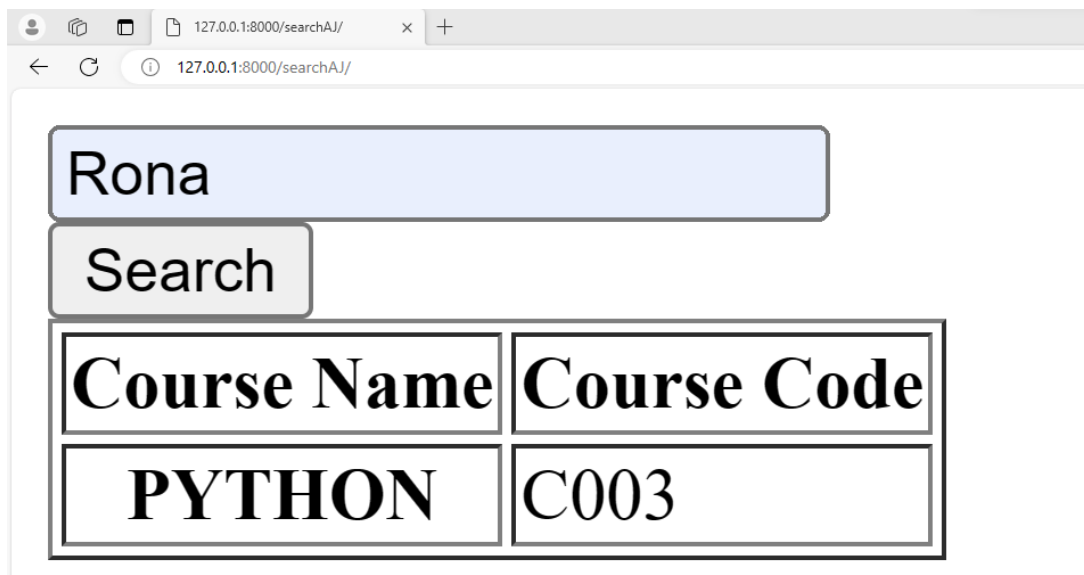
A screenshot of a web browser window. The address bar shows '127.0.0.1:8000/searchAJ/'. The page content consists of a large text input field with the placeholder text 'Student Name' and a button labeled 'Search' below it.

On clicking Search button,



A screenshot of a web browser window showing the search results for 'Sona'. The input field contains 'Sona', and the 'Search' button is visible. Below the button, a table displays course information:

Course Name	Course Code
PYTHON	C003
JAVA	C005



A screenshot of a web browser window showing the search results for 'Rona'. The input field contains 'Rona', and the 'Search' button is visible. Below the button, a table displays course information:

Course Name	Course Code
PYTHON	C003