

Parul University

FACULTY OF ENGINEERING AND TECHNOLOGY

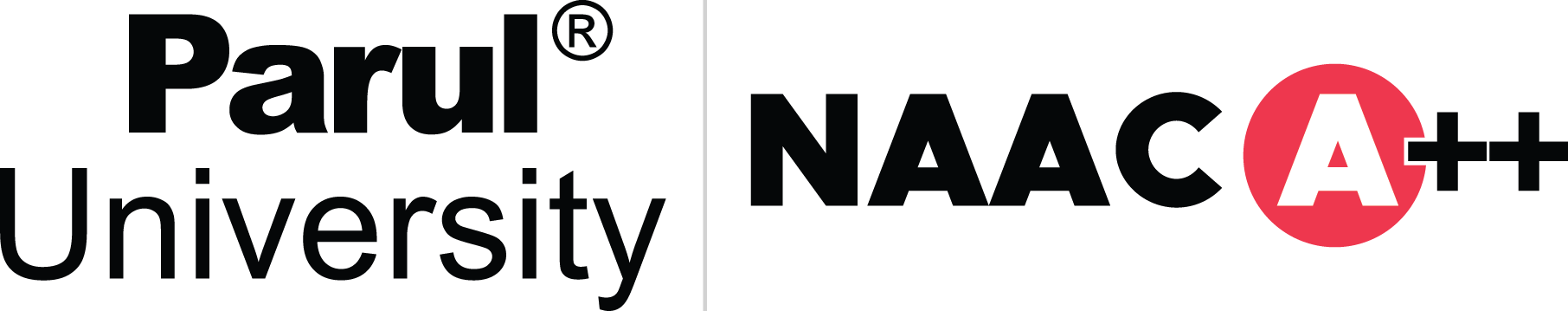
BACHELOR OF TECHNOLOGY

PROGRAMMING IN PYTHON WITH FULL STACK DEVELOPMENT

(303105258)

IV SEMESTER

Computer Science & Engineering Department



Laboratory Manual

Session 2023-24

CERTIFICATE

This is to Certify that

Mr./Ms. ㅤㅤㅤㅤㅤ SHIVAM SINGH ㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤ

With enrolment no. ㅤ 2203051050534ㅤ ㅤhas successfully completed his/her

Laboratory experiments in **Python Full Stack Development (303105258)** From the

department of ㅤㅤㅤCOMPUTER SCIENCE AND ENGINEERING ㅤ ㅤㅤduring the academic year 2023 - 2024ㅤㅤ



Date of Submission : ………………………. Staff In Charge: ……………………….

Head of department: ……………………….

**TABLE OF CONTENT**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Experimental Title** | **Page No** | | **Date of Completion** | **Sign** | **Marks (out of 10)** |
| **From** | **To** |
| **SET 1** | | | | | | |
| 1. | A program that converts temperatures from Fahrenheit to Celsius and vice versa. |  |  |  |  |  |
| 2. | A program that calculates the area and perimeter of a rectangle. |  |  |  |  |  |
| 3. | A program that generates a random password of a specified length |  |  |  |  |  |
| 4. | A program that calculates the average of a list of numbers. |  |  |  |  |  |
| 5. | A program that checks if a given year is a leap year. |  |  |  |  |  |
| 6. | A program that calculates the factorial of a number. |  |  |  |  |  |
| 7. | A program that checks if a given string is a palindrome. |  |  |  |  |  |
| 8. | A program that sorts a list of numbers in ascending or descending order. |  |  |  |  |  |
| 9. | A program that generates a multiplication table for a given number. |  |  |  |  |  |
| 10. | A program that converts a given number from one base to another. |  |  |  |  |  |
| **SET 2** | | | | | | |
| 1. | A program that models a bank account, with classes for the account, the customer,  and the bank. |  |  |  |  |  |
| 2. | A program that simulates a school management system, with classes for the  students, the teachers, and the courses. |  |  |  |  |  |
| 3. | A program that reads a text file and counts the number of words in it |  |  |  |  |  |
| 4. | A program that reads a CSV file and calculates the average of the values in a specified column. |  |  |  |  |  |
| 5. | A program that reads an Excel file and prints the data in a tabular format. |  |  |  |  |  |
| **SET 3** | | | | | | |
| 1. | A program that creates a simple web server and serves a static HTML page |  |  |  |  |  |
| 2. | A program that creates a web application that allows users to register and login. |  |  |  |  |  |
| 3. | A program that creates a web application that allows users to upload and download files. |  |  |  |  |  |
| 4. | A program that creates a web application that displays data from a database in a tabular format |  |  |  |  |  |
| 5. | A program that creates a web application that accepts user input and sends it to a server-side script for processing. |  |  |  |  |  |
| **SET 4** | | | | | | |
| 1. | A program that creates a web application that uses a template engine to generate dynamicHTML pages. |  |  |  |  |  |
| 2. | A program that creates a web application that supports AJAX requests and updates the page without reloading. |  |  |  |  |  |
| 3. | A program that creates a web application that uses Django's built-in debugging features to troubleshoot errors and  exceptions. |  |  |  |  |  |
| 4. | A program that creates a web application that implements user authentication and authorization. |  |  |  |  |  |
| 5. | A program that creates a web application that integrates with third-party APIs to provide additional functionality. |  |  |  |  |  |
| **SET 5** | | | | | | |
| 1. | A program that creates a simple RESTful API that returns a list of users in JSON format. |  |  |  |  |  |
| 2. | A program that creates a RESTful API that allows users to create, read, update, and delete resources. |  |  |  |  |  |
| 3. | A program that creates a RESTful API that authenticates users using a JSON Web Token. |  |  |  |  |  |
| 4. | A program that creates a RESTful API that paginates the results of a query to improve performance. |  |  |  |  |  |
| 5. | A program that creates a RESTful API that supports data validation and error handling. |  |  |  |  |  |

# **PRACTICAL - 1**

AIM - A program that convert tempratures from Fahrenheit to Celsius and vice versa.

Program –

def celsius\_to\_fahrenheit(celsius):

    fahrenheit = (celsius \* 9/5) + 32

    return fahrenheit

def fahrenheit\_to\_celsius(fahrenheit):

    celsius = (fahrenheit - 32) \* 5/9

    return celsius

celsius = 25

fahrenheit = celsius\_to\_fahrenheit(celsius)

print(f"{celsius} degrees Celsius is equal to {fahrenheit} degrees Fahrenheit.")

fahrenheit = 77

celsius = fahrenheit\_to\_celsius(fahrenheit)

print(f"{fahrenheit} degrees Fahrenheit is equal to {celsius} degrees Celsius.")

OUTPUT:

25 degrees Celsius is equal to 77.0 degrees Fahrenheit.

77 degrees Fahrenheit is equal to 25.0 degrees Celsius.

# **PRACTICAL – 2**

AIM- A program that calculate the area and perimeter of rectangle

Program –

def calculate\_area(length, width):

    return length \* width

def calculate\_perimeter(length, width):

    return 2 \* (length + width)

length = 5

width = 3

area = calculate\_area(length, width)

perimeter = calculate\_perimeter(length, width)

print("Area is :", area)

print("Perimeter is :", perimeter)

OUTPUT:-

Area is : 15

Perimeter is : 16

# **PRACTICAL – 3**

AIM - A program that generates a random password of a specified length.

Program –

import random

import string

def ranpass(length):

    passs = string.digits + string.ascii\_letters

    result = ''.join(random.choice(passs) for i in range(length))

    print("randomly generated password is:", result)

ranpass(8)

OUTPUT:-

randomly generated password is: TJpF52zE

# **PRACTICAL – 4**

AIM - A program that calculate the average of a list of numbers.

Program –

n = int(input("Enter the number of elements in the list: "))

lst = []

for i in range(n):

    lst.append(int(input("Enter element: ")))

average = sum(lst) / len(lst)

print("The average of the list is", average)

OUTPUT:-

Enter the number of elements in the list: 4

Enter element: 23

Enter element: 2322

Enter element: 34534

Enter element: 23423

The average of the list is 15075.5

# **PRACTICAL – 5**

AIM - A program that chek if a given year is leap year

Program –

def is\_leap\_year(year):

    if year % 4 == 0:

        if year % 100 == 0:

            if year % 400 == 0:

                return True

            else:

                return False

        else:

            return True

    else:

        return False

year = int(input("Enter a year: "))

if is\_leap\_year(year):

    print(year, "is a leap year")

else:

    print(year, "is not a leap year")

OUTPUT:-

Enter a year: 2002

2002 is not a leap year

# **PRACTICAL – 6**

AIM - A program that calculates the factorial of a number.

Program –

n = int(input("Enter a number:"))

fact = 1

if n == 0:

    print('1')

elif n < 0:

    print("not define")

else:

    for i in range(1, n+1):

        fact = i \* fact

print("factorial of given number is", fact)

OUTPUT:-

Enter a number:5

factorial of given number is 120

# **PRACTICAL – 7**

AIM - A program that checks if a given string is a palindrome

Program –

str1 = input("enter the string to check palindrome: ")

str2 = ''.join(reversed(str1))

if str1 == str2:

    print("palindrome")

else:

    print("not palindrome")

OUTPUT:-

enter the string to check palindrome: 121

121 is palindrome

enter the string to check palindrome: 112

112 not palindrome

enter the string to check palindrome: madam

madam is palindrome

enter the string to check palindrome: engineer

engineer not palindrome

# **PRACTICAL – 8**

AIM - A program that sorts a list of numbers in ascending or descending order.

Program –

s = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

s.sort()

print(f"The sorted list is: {s}")

s.sort(reverse=True)

print(f"The reverse sorted list is: {s}")

listt = ["apple", "banana", "cherry", "date"]

listt.sort()

print(f"The sorted list is: {listt}")

listt.sort(reverse=True)

print(f"The reverse sorted list is: {listt}")

OUTPUT:-

The sorted list is: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

The reverse sorted list is: [9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]

The sorted list is: ['apple', 'banana', 'cherry', 'date']

The reverse sorted list is: ['date', 'cherry', 'banana', 'apple']

# **PRACTICAL – 9**

AIM - A program that generates a multiplication table for a given number.

Program –

def table(n):

    print("the table for given number are below: ")

    for i in range(1, 11, 1):

        print("5 \*", i, "=", (n \* i))

table(5)

OUTPUT:-

the table for given number are below:

5 \* 1 = 5

5 \* 2 = 10

5 \* 3 = 15

5 \* 4 = 20

5 \* 5 = 25

5 \* 6 = 30

5 \* 7 = 35

5 \* 8 = 40

5 \* 9 = 45

5 \* 10 = 50

# **PRACTICAL – 10**

AIM - A program that converts a given number from one base to another

Program –

def convert\_to\_all\_formats(number):

    print("the given number in all other format are given below")

    hexa = hex(number)

    octa = oct(number)

    binary = bin(number)

    print(f"in Hexadecimal: {hexa}")

    print(f"in Octal: {octa}")

    print(f"in Binary: {binary}")

convert\_to\_all\_formats(42)

OUTPUT:-

the given number in all other format are given below

in Hexadecimal: 0x2a

in Octal: 0o52

in Binary: 0b101010

# **PRACTICAL – 1**

AIM - A program that models a bank account, with classes for the account, the customer, and the bank.

Program –

class BankAccount:

    def \_\_init\_\_(self):

        self.balance = 0

    def deposit(self):

        amount = int(input("Enter amount to be deposited: "))

        self.balance += amount

        print(f"Amount Deposited: ${amount}")

    def withdraw(self):

        amount = int(input("Enter amount to be withdrawn: "))

        if self.balance >= amount:

            self.balance -= amount

            print(f"You Withdrew: ${amount}")

        else:

            print("Insufficient balance")

    def display\_balance(self):

        print(f"Net Available Balance: ${self.balance}")

account = BankAccount()

account.deposit()

account.withdraw()

account.display\_balance()

OUTPUT:-

Enter amount to be deposited: 40

Amount Deposited: $40

Enter amount to be withdrawn: 10

You Withdrew: $10

Net Available Balance: $30

# **PRACTICAL – 2**

AIM - A program that simulates a school management system, with classes for the students, the teachers, and the courses.

Program –

class Course:

def \_\_init\_\_(self, course\_id, course\_name):

self.course\_id = course\_id

self.course\_name = course\_name

self.teacher = None

self.students = []

def add\_student(self, student):

self.students.append(student)

def set\_teacher(self, teacher):

self.teacher = teacher

class Student:

def \_\_init\_\_(self, student\_id, name, grade):

self.student\_id = student\_id

self.name = name

self.grade = grade

self.courses = []

def enroll(self, course):

self.courses.append(course)

course.add\_student(self)

def display\_courses(self):

print(f"{self.name} is enrolled in the following courses:")

for course in self.courses:

print(course.course\_name)

class Teacher:

def \_\_init\_\_(self, teacher\_id, name, subject):

self.teacher\_id = teacher\_id

self.name = name

self.subject = subject

self.courses = []

def assign\_course(self, course):

self.courses.append(course)

course.set\_teacher(self)

def display\_courses(self):

print(f"{self.name} is assigned to teach the following courses:")

for course in self.courses:

print(course.course\_name)

course1 = Course(1, "Java")

course2 = Course(2, "Python")

student1 = Student(1, "ENGINEER", 10)

student2 = Student(2, "Ravi", 20)

teacher1 = Teacher(101, "Shivam", "Java")

teacher2 = Teacher(102, "Gobinda", "Python")

teacher1.assign\_course(course1)

teacher2.assign\_course(course2)

student1.enroll(course1)

student2.enroll(course2)

OUTPUT:-

ENGINEER is enrolled in the following courses:

Java

Shivam is assigned to teach the following courses:

Java

Ravi is enrolled in the following courses:

Python

Gobinda is assigned to teach the following courses:

Python

student1.display\_courses()

teacher1.display\_courses()

student2.display\_courses()

teacher2.display\_courses()

# **PRACTICAL – 3**

AIM: A program that reads a text file and counts the number of words in it.

Program –

file\_path = "C:\\Users\\babue\Desktop\\test.txt.txt"

with open(file\_path, 'r') as file:

# Read the contents of the file

file\_content = file.read()

# Print the text

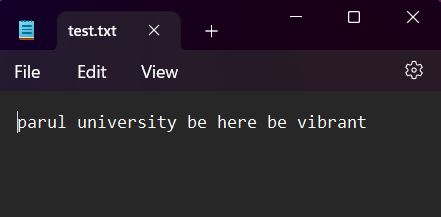
print(file\_content)

data = file\_content.split()

length = len(file\_content.split())

print(f"content in file = {data} \nlengh of the content = {length}")

TEXT FILE:



**OUTPUT:**

parul university be here be vibrant

content in file = ['parul', 'university', 'be', 'here', 'be', 'vibrant']

lengh of the content = 6

# **PRACTICAL – 4**

AIM: A program that reads a CSV file and calculates the average of the values in a specified column

Program –

import csv

def calculate\_average(csv\_file, column\_name):

try:

with open(csv\_file, 'r') as file:

reader = csv.DictReader(file)

if column\_name not in reader.fieldnames:

print(f"Column '{column\_name}' not found in the CSV file.")

return None

total = 0

count = 0

for row in reader:

try:

value = float(row[column\_name])

total += value

count += 1

except ValueError:

print(f"Skipping row {reader.line\_num}: Invalid value in column '{column\_name}'.")

if count == 0:

print(f"No valid values found in column '{column\_name}'.")

return None

average = total / count

return average

except FileNotFoundError:

print(f"File '{csv\_file}' not found.")

return None

csv\_file\_path = 'test.csv'

column\_to\_calculate = 'PFSD'

result = calculate\_average(csv\_file\_path, column\_to\_calculate)

if result is not None:

print(f"The average value in column '{column\_to\_calculate}' is: {result}")

CSV FILE DATA:

A screenshot of a computer

Description automatically generated

**OUTPUT:**

**The average value in column 'PFSD' is: 50.36363636363637**

# **PRACTICAL – 5**

AIM: A program that reads an Excel file and prints the data in a tabular format.

Program –

**import pandas as pd**

**import openpyxl**

**output = pd.read\_excel("test.xlsx")**

**print(output)**

EXCEL FILE DATA:

A screenshot of a computer

Description automatically generated

**OUTPUT:**

A screenshot of a computer screen

Description automatically generated

# **PRACTICAL – 1**

AIM : A program that creates a simple web server and serves a static HTML page.

Implementation:

Create a new folder structure as shown in picture

Create a new folder with name “PY”

A screenshot of a computer

Description automatically generated

Step 1. Open PY folder in VS code

Step 2. Create subfolder with name templates

Step 3. Inside template folder create file index.html

Step 4. Create python file with name test,py in main folder

**IMPORTANT:**

In terminal type the command “pip install Flask” to install flask on your system

**Content of index.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>Static HTML Page</title>

<style>

body {

background-color: #f2f2f2;

font-family: Arial, sans-serif;

}

h1 {

color: #007bff;

text-align: center;

margin-top: 50px;

}

</style>

</head>

<body>

<h1>ENGINEER</h1>

</body>

</html>

**Content of test.py file:**

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**OUTPUT:**

**A black background with red text

Description automatically generated**

navigate to http://127.0.0.1:5000/ in a web browser. You should see a static HTML page with the expected content

**A screenshot of a computer

Description automatically generated**

# **PRACTICAL – 2**

❖ AIM: A program that creates a web application that allows users to register and login.

Setup:

In terminal type the command “pip install Flask Flask-SQLAlchemy” to install flask on your system

Create a new folder with name “PY”

Open PY folder in VS code

Create Project Structure as shown in picture:

**A screenshot of a computer

Description automatically generated**

Step 1. Open PY folder in VS code

Step 2. Create subfolder with name templates

Step 3. Inside template folder create file index.html

Step 4. Inside template folder create file login.html

Step 5. Inside template folder create file register.html

Step 6. Create python file with name test,py in main folder

**Content of test.py file:**

from flask import Flask, render\_template, request, redirect, url\_for, session, flash

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash

import secrets

app = Flask(\_\_name\_\_)

app.secret\_key = secrets.token\_hex(16)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///users.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(50), unique=True, nullable=False)

password = db.Column(db.String(256), nullable=False)

with app.app\_context():

db.create\_all()

@app.route("/")

def home():

return render\_template("index.html")

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if User.query.filter\_by(username=username).first():

flash('Username already taken. Please choose another.', 'error')

else:

hashed\_password = generate\_password\_hash(password, method='pbkdf2:sha256')

new\_user = User(username=username, password=hashed\_password)

db.session.add(new\_user)

db.session.commit()

flash('Registration successful. You can now log in.', 'success')

return redirect(url\_for('login'))

return render\_template('register.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

if user and check\_password\_hash(user.password, password):

session['username'] = username

flash('Login successful!', 'success')

return redirect(url\_for('dashboard'))

else:

flash('Invalid username or password. Please try again.', 'error')

return render\_template('login.html')

@app.route('/dashboard')

def dashboard():

if 'username' in session:

return f'Welcome to the dashboard, {session["username"]}!'

else:

flash('Please log in to access the dashboard.', 'info')

return redirect(url\_for('login'))

@app.route('/logout')

def logout():

session.pop('username', None)

flash('You have been logged out.', 'info')

return redirect(url\_for('login'))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Content of index.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

<title>Static HTML Page</title>

</head>

<style>

@import url("https://fonts.googleapis.com/css2?family=Poppins:wght@500&display=swap");

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

height: 100vh;

width: 100%;

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

background: #ff5a5f;

}

h1 {

font-family: "Poppins", sans-serif;

color: #fff;

margin: 30px 50px;

font-size: 3rem;

}

input {

padding: 10px 20px;

border: 3px solid #fff;

border-radius: 10px;

background: rgb(16, 208, 16);

font-size: 1.5rem;

color: white;

font-family: "Poppins", sans-serif;

font-weight: 300;

transition: .3s;

&:hover{

background: #fff;

color: #000;

cursor: pointer;

}

}

</style>

<body>

<h1>Hello, this is a static HTML page served by Flask!</h1>

<form action="{{ url\_for('register') }}">

<input type="submit" value="Register" />

</form>

</body>

</html>

**Content of register.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

<title>User Registration</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

height: 100vh;

width: 100%;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

background: rgb(9, 9, 121);

background: linear-gradient(

30deg,

rgba(9, 9, 121, 1) 0%,

rgba(2, 0, 36, 1) 29%,

rgba(0, 212, 255, 1) 100%

);

}

.container {

display: flex;

align-items: center;

justify-content: space-evenly;

flex-direction: column;

width: 600px;

border-radius: 20px;

height: 500px;

background: #ffffff5a;

backdrop-filter: blur(20px);

& h1 {

font-family: Arial, Helvetica, sans-serif;

color: #fff;

margin: 30px 0;

}

& li {

list-style: none;

}

& form {

& label {

color: white;

font-family: Arial, Helvetica, sans-serif;

font-size: 1.4rem;

margin: 10px 20px;

}

& .register\_button {

color: #fff;

background: red;

border: none;

outline: none;

padding: 5px 10px;

border-radius: 10px;

font-size: 1.2rem;

transition: 0.3s;

transform: translateX(130px);

&:hover {

background: #fff;

color: #000;

cursor: pointer;

}

}

& .password {

padding: 10px 20px;

border-radius: 20px;

outline: none;

border: none;

}

& .username {

padding: 10px 20px;

border-radius: 20px;

outline: none;

border: none;

}

& input {

margin: 10px 20px;

}

}

}

.error {

color: red;

}

.success {

color: green;

}

.default {

color: black;

}

</style>

</head>

<body>

<div class="container">

<h1>User Registration</h1>

{% with messages = get\_flashed\_messages() %} {% if messages %}

<ul>

{% for message in messages %}

<li

class="{% if 'error' in message %}error{% elif 'success' in message %}success{% else %}default{% endif

%}"

>

{{ message }}

</li>

{% endfor %}

</ul>

{% endif %} {% endwith %}

<form method="post" action="{{ url\_for('register') }}">

<label for="username" class="username\_label">Username:</label>

<input type="text" name="username" class="username" required />

<br />

<label for="password" class="password\_label">Password:</label>

<input type="password" name="password" class="password" required />

<br />

<input type="submit" class="register\_button" value="Register" />

</form>

<p>

Already have an account?

<a href="{{ url\_for('login') }}">Log in here</a>.

</p>

</div>

</body>

</html>

**OUTPUT:**

**A black screen with red text

Description automatically generated**

navigate to [http://127.0.0.1:5000/](http://127.0.0.1:5000/%20) in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# **PRACTICAL – 3**

AIM: A program that creates a web application that allows users to upload and download files.

Procedure:

Setup:

1. • Make sure you have Python installed on your system.
2. • Install Flask: bash pip install Flask

Structure of Folder:

A screenshot of a computer

Description automatically generated

**Content of test.py file:**

from flask import Flask, render\_template, request, send\_from\_directory, redirect, url\_for

import os

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

@app.route('/')

def index():

filenames = os.listdir(app.config['UPLOAD\_FOLDER'])

return render\_template('index.html', filenames=filenames)

@app.route('/upload', methods=['POST'])

def upload\_file():

if 'file' not in request.files:

return "No file part"

file = request.files['file']

if file.filename == '':

return "No selected file"

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], file.filename))

return redirect(url\_for('index'))

@app.route('/download/<filename>')

def download\_file(filename):

return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Content of index.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>File Upload and Download</title>

</head>

<body>

<h1>File Upload and Download</h1>

<form action="/upload" method="post" enctype="multipart/form-data">

<label for="file">Choose a file:</label>

<input type="file" name="file" id="file" required>

<br>

<input type="submit" value="Upload">

</form>

<h2>Uploaded Files</h2>

{% for filename in filenames %}

<div>

<span>{{ filename }}</span>

<a href="{{ url\_for('download\_file', filename=filename) }}" download>

<button>Download</button>

</a>

</div>

{% endfor %}

</body>

</html>

**OUTPUT:**

A black screen with red text

Description automatically generated

navigate to [http://127.0.0.1:5000/](http://127.0.0.1:5000/%20) in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# **PRACTICAL – 4**

AIM: A program that creates a web application that displays data from a database in a tabular format.

Procedure:

Setup:

• Ensure you have Python installed on your system.

• Install Flask and SQLAlchemy: bash pip install Flask Flask-SQLAlchemy pandas

Structure of Folder:

A screenshot of a computer

Description automatically generated

**Content of test.py file:**

from flask import Flask, render\_template

from flask\_sqlalchemy import SQLAlchemy

import pandas as pd

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///example.db'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

# Create a SQLAlchemy instance

db = SQLAlchemy(app)

# Define a model for the data

class Person(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(50), nullable=False)

age = db.Column(db.Integer, nullable=False)

# Sample data for demonstration

sample\_data = [{'name': 'RAVI', 'age': 20},

{'name': 'ENGINEER', 'age': 20},

{'name': 'SHIVAM', 'age': 20}]

# Populate the database with sample data

with app.app\_context():

db.create\_all()

for entry in sample\_data:

person = Person(name=entry['name'], age=entry['age'])

db.session.add(person)

db.session.commit()

# Define a route to display data in tabular format

@app.route('/')

def display\_data():

# Query data from the database

data = Person.query.all()

# Convert the data to a Pandas DataFrame

df = pd.DataFrame([(person.name, person.age) for person in data], columns=['name', 'age'])

# Convert the DataFrame to HTML for rendering in the template

table\_html = df.to\_html(classes='table table-striped', index=False)

return render\_template('index.html', table\_html=table\_html)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Content of index.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Data Display</title>

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h1>Data Display <small class="text-right">ENGINEER</small></h1>

<!-- Render the HTML table -->

{{ table\_html | safe }}

</div>

</body>

</html>

**OUTPUT:**

A black screen with red text

Description automatically generated

navigate to [http://127.0.0.1:5000/](http://127.0.0.1:5000/%20) in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated

# **PRACTICAL – 5**

**AIM:** A program that creates a web application that accepts user input and sends it to a server-side script for processing.

**Procedure:**

Setup:

1. • Make sure you have Python installed on your system.
2. • Install Flask: bash pip install Flask

Structure of Folder:

A screenshot of a computer

Description automatically generated

**Content of test.py file:**

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

# Define a route for the main page

@app.route('/', methods=['GET', 'POST'])

def index():

result = None

if request.method == 'POST':

# Get user input from the form

user\_input = request.form.get('user\_input')

result = f"You entered: {user\_input}"

return render\_template('index.html', result=result)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Content of index.html file:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>User Input</title>

</head>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

height: 100vh;

width: 100%;

background: #a2d2ff;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

}

.container {

display: flex;

align-items: center;

justify-content: space-evenly;

flex-direction: column;

width: 500px;

height: 600px;

border-radius: 20px;

background: #ffffff5a;

backdrop-filter: blur(20px);

& h1{

font-family: Arial, Helvetica, sans-serif;

color: #3a86ff;

font-size: 2rem;

}

& label{

color: #3a86ff;

font-family: Arial, Helvetica, sans-serif;

font-size: 1.2rem;

padding: 10px;

margin: 10px 20px;

}

& .enter{

padding: 10px 20px;

border: none;

outline: none;

border-radius: 20px;

}

& .submit{

padding: 10px 20px;

color: #fff;

background: #2a9d8f;

outline: none;

border: none;

border-radius: 10px;

transition: .3s;

transform: translateX(150px);

margin: 30px;

&:hover{

color: #000;

cursor: pointer;

background: #fff;

}

}

& h2{

font-family: Arial, Helvetica, sans-serif;

color: #3a86ff;

font-size: 2rem;

}

}

</style>

<body>

<div class="container">

<h1>User Input Form</h1>

<form method="post" action="/">

<label for="user\_input">Enter something:</label>

<input type="text" class="enter" name="user\_input" id="user\_input" required />

<br />

<input class="submit" type="submit" value="Submit" />

</form>

{% if result %}

<div>

<h2>Result:</h2>

<p>{{ result }}</p>

</div>

{% endif %}

</div>

</body>

</html>

**OUTPUT:**

A black screen with red text

Description automatically generated

navigate to [http://127.0.0.1:5000/](http://127.0.0.1:5000/%20) in a web browser. You should see a static HTML page with the expected content

A screenshot of a login form

Description automatically generated A screenshot of a user input form

Description automatically generated

# **PRACTICAL – 1**

**AIM :** A program that creates a web application that uses a template engine to generate dynamic HTML pages.

**Procedure:**

1)Setup:

* Make sure you have Python installed on your system.
* Create new empty folder(PY) and open in vs code
* In terminal type the Below command
* “pip install Django”
* “django-admin startproject myproject”
* “cd myproject”
* “python manage.py startapp myapp”

2) Configuring project-level url routing: 2 • Add a path inside (myproject/myproject/urls.py) using include function from django.urls.

**Urls.py :**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

3) Defining the Application: • Add the application as “myapp” inside the INSTALLED\_APPS variable inside (myproject/myproject/settings.py).

Search for commented line application definition and modify that section according to given lines below

**Settings.py :**

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

"myapp"

]

4) Defining models: • Define models in models.py file within the app directory (myapp/models.py).

**Models.py :**

from django.db import models

class MyModel(models.Model):

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

# Add more fields as needed

5) Defining views: • Define views in views.py file within the app directory (myapp/views.py).

**Views.py :**

from django.shortcuts import render

from .models import MyModel

def my\_view(request):

data = MyModel.objects.all()

return render(request, 'my\_template.html', {'data': data})

6) Configuring URL routing: • Define URL patterns in urls.py file within the app directory (myapp/urls.py).

If not available create file urls.py inside app directory

**Urls.py :**

from django.urls import path

from . import views

urlpatterns = [

path('', views.my\_view, name='my\_view'),

# Add more URL patterns as needed

]

7) Configuring URL routing: • Create HTML templates in templates directory within the app directory (myapp/templates).

**my\_templates.html :**

<html>

<head>

<title>Dynamic Page</title>

</head>

<body>

<h1>Dynamic Content</h1>

<ul>

{% for item in data %}

<li>{{ item.name }}</li>

{% endfor %}

</ul>

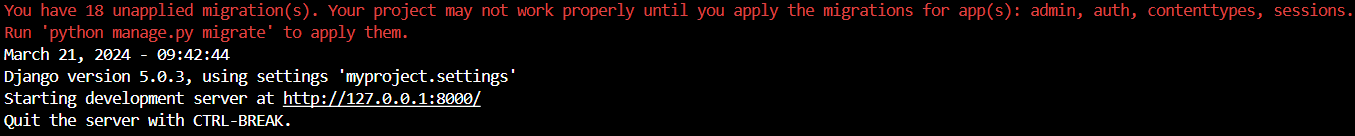
</body>

</html>

8) Running the server:

• Start the Django development server: by typing cmd in terminal “ python manage.py runserver “

**Expected Output:**

****

navigate to <http://127.0.0.1:8000/> in a web browser. You should see a static HTML page with the expected content

A white screen with black text

Description automatically generated

# **PRACTICAL – 2**

**AIM:** A program that creates a web application that supports AJAX requests and updates the page without reloading.

**Setting up Django project:**

1. pip install Django
2. django-admin startproject myproject
3. cd myproject
4. python manage.py startapp myapp

**Configuring Project-level url routing:**

Add a path inside (myproject/myproject/urls.py) using include function from django.urls.

**urls.py :**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

**Defining the Application:**

• Add the application as “myapp” inside the INSTALLED\_APPS variable inside (myproject/myproject/settings.py).

**settings.py :**

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

"myapp"

]

**Defining views for AJAX requests:**

• Define views in views.py file within the app directory (myapp/views.py).

**views.py :**

from django.http import JsonResponse

def ajax\_update(request):

# Process AJAX request and prepare data

data = {'message': 'Updated content from AJAX request!'}

return JsonResponse(data)

**Configuring URL routing:**

• Define URL patterns in urls.py file within the app directory (myapp/urls.py).

**urls.py :**

from django.urls import path

from . import views

urlpatterns = [

path('ajax\_update/', views.ajax\_update, name='ajax\_update'),

# Add more URL patterns as needed

]

**HTML Structure with a Placeholder for Dynamic Content and JavaScript function for AJAX request:**

Create an HTML file with a placeholder for the dynamic content and write JavaScript code to send AJAX request and update the page content dynamically.

**index.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

scale=1.0">

<title>AJAX Example</title>

</head>

<body>

<h1>Dynamic Content Update</h1>

<div id="content">Placeholder for dynamic content</div>

<button onclick="updateContent()">Update Content</button>

<script>

function updateContent() {

fetch('/myapp/ajax\_update/')

.then(response => response.json())

.then(data => {

// Update DOM with new content

document.getElementById('content').innerHTML =

data.message;

})

.catch(error => console.error('Error:', error));

}

</script>

</body>

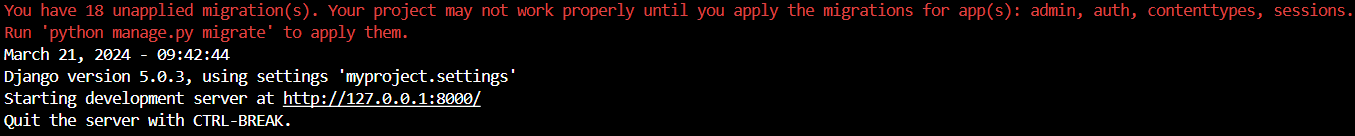
</html>

**Running the server:**

• Start the Django development server:

“python manage.py runserver”

**Expected Output:**

****

navigate to <http://127.0.0.1:8000/> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated

# **PRACTICAL – 3**

**AIM:** A program that creates a web application that uses Django's built-in debugging features to troubleshoot errors and exceptions.

**1) Setting up Django project:**

pip install django

django-admin startproject myproject

cd myproject

python manage.py startapp myapp

**2) Enable Debugging in Django settings:**

• Ensure that Django's debug mode is enabled in the project's settings file

(myproject/settings.py):

settings.py

DEBUG = True

**3) Creating a view for the calculator:**

• Define a view in myapp/views.py to handle user input and calculate the square:

**Views.py :**

from django.shortcuts import render

from django.http import HttpResponse

def calculate\_square(request):

try:

if request.method == 'POST':

number = int(request.POST.get('number'))

result = number \*\* 2

return HttpResponse(f"The square of {number} is

{result}.")

except Exception as e:

return HttpResponse(f"An error occurred: {e}")

return render(request, 'index.html')

**4) Creating HTML template for the calculator:**

• Create an HTML template index.html in calculator/templates directory:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Calculator</title>

</head>

<body>

<h1>Calculate Square</h1>

<form method="post" action="{% url 'calculate\_square'

%}">

{% csrf\_token %}

<label for="number">Enter a number:</label>

<input type="number" id="number" name="number"

required>

<button type="submit">Calculate Square</button>

</form>

</body>

</html>

**5)Configuring URL routing:**

• Define URL patterns in urls.py file within the app directory (myapp/urls.py). Example:

**urls.py :**

from django.urls import path

from . import views

urlpatterns = [

path('', views.calculator\_square, name=calculator\_square'),

# Add more URL patterns as needed

]

**6)Configuring Project-level url routing:**

• Add a path inside (myproject/myproject/urls.py) using include function from django.urls.

Example:

urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

**7) Running the server:**

• Start the Django development server:

python manage.py runserver

**Expected Output:**

****

navigate to <http://127.0.0.1:8000/> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

# **PRACTICAL – 4**

**AIM:** Develop a Python program to create a web application using Django framework that implements user authentication and authorization to control access to resources based on user roles and permissions.

**1) Setting up Django project:**

pip install django

django-admin startproject myproject

cd myproject

**2) Creating Django app:**

python manage.py startapp myapp

**3) Configuring Project-level url routing:**

• Add a path inside (myproject/myproject/urls.py) using include function from django.urls.

**urls.py :**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

**4) Defining the Application:**

• Add the application as “myapp” inside the INSTALLED\_APPS variable inside

(myproject/myproject/settings.py).

**settings.py :**

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

"myapp"

]

**5) Configuring User Authentication:**

• Django provides a built-in authentication system that includes user models, forms, views, and

authentication backends.

• Customize the user model in myapp/models.py if needed to add additional fields or

functionality:

**models.py :**

from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):

# Add custom fields as needed

Pass

**6) Implementing User Registration and Login:**

• Create views and templates for user registration, login, logout, and password reset as needed.

• Views:

**views.py :**

# myapp/views.py

from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.forms import UserCreationForm,

AuthenticationForm

from django.contrib.auth.views import PasswordResetView

from django.shortcuts import render, redirect

def register(request):

if request.method == 'POST':

form = UserCreationForm(request.POST)

if form.is\_valid():

form.save()

return redirect('login')

else:

form = UserCreationForm()

return render(request, 'registration/register.html',

{'form': form})

def user\_login(request):

if request.method == 'POST':

form = AuthenticationForm(request, request.POST)

if form.is\_valid():

username = form.cleaned\_data.get('username')

password = form.cleaned\_data.get('password')

user = authenticate(username=username,

password=password)

if user is not None:

login(request, user)

return redirect('home')

else:

form = AuthenticationForm()

return render(request, 'registration/login.html', {'form':

form})

def user\_logout(request):

logout(request)

return redirect('login')

**• Templates:**

**registration/register.html :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Register</title>

</head>

<body>

<h1>Register</h1>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Register</button>

</form>

</body>

</html>

registration/register.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login</title>

</head>

<body>

<h1>Login</h1>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Login</button>

</form>

</body>

</html>

**• Customize the user model in myapp/models.py if needed to add additional fields or**

**functionality:**

**models.py :**

from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):

# Add custom fields as needed

Pass

**7) Url Configuration:**

**urls.py :**

from django.urls import path

from . import views

urlpatterns = [

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

path('login/', views.user\_login, name='login'),

path('logout/', views.user\_logout, name='logout'),

# Add more URL patterns as needed

]

**8) Running the server:**

python manage.py runserver

**Expected Output:**

****

navigate to <http://127.0.0.1:8000/> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer screen

Description automatically generated A screenshot of a computer

Description automatically generated

# **PRACTICAL – 5**

**AIM:** A program that creates a web application that integrates with third party APIs to provide additional functionality.

**1) Setting up Django project:**

pip install django

django-admin startproject myproject

cd myproject

**2) Creating Django app:**

python manage.py startapp myapp

**3) Configuring Project-level url routing:**

• Add a path inside (myproject/myproject/urls.py) using include function from django.urls.

**urls.py :**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

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

]

**4) Defining the Application:**

• Add the application as “myapp” inside the INSTALLED\_APPS variable inside

(myproject/myproject/settings.py).

**settings.py :**

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

"myapp"

]

**5) Create Functions for API Integration:**

• First, you'll create a function in your Django app (myapp) to interact with the Public APIs

API.

**utils.py :**

import requests

def get\_public\_apis():

url = "https://api.publicapis.org/entries"

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

**6) Implement Views to Display API Data:**

• Next, you'll create a view in myapp/views.py to render data obtained from the Public

APIs API.

**views.py :**

from django.shortcuts import render

from .utils import get\_public\_apis

def public\_apis\_view(request):

api\_data = get\_public\_apis()

return render(request, ‘index.html', {'api\_data':

api\_data})

**7) Create HTML Templates:**

• Then, you'll develop an HTML template in the myapp/templates directory to display the

data fetched from the Public APIs API.

**index.html :**

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Public APIs</title>

</head>

<body>

<h1>Public APIs</h1>

<ul>

{% for entry in api\_data.entries %}

<li>{{ entry.API }} - {{ entry.Description

}}</li>

{% endfor %}

</ul>

</body>

</html>

**8) Configure URL Routing:**

• Define a URL pattern in myapp/urls.py to map the view to a URL endpoint.

**urls.py :**

from django.urls import path

from . import views

urlpatterns = [

path('public-apis/', views.public\_apis\_view,

name='public\_apis'),

]

**9) Running the server:**

python manage.py runserver

**Expected Output:**

****

navigate to <http://127.0.0.1:8000/public-apis/> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated

# **PRACTICAL – 1**

**AIM:** A program that creates a simple RESTful API that returns a list of users in JSON format

**Procedure:**

**Step 1:**

Write the Flask App Open app.py in a text editor and write the Flask application code:

**app.py:**

from flask import Flask, jsonify

app = Flask( name )

users = [

{'id': 1, 'name': 'ENGINEER'},

{'id': 2, 'name': 'RAVI'},

{'id': 3, 'name': 'SHIVAM'}

]

@app.route('/users', methods=['GET'])

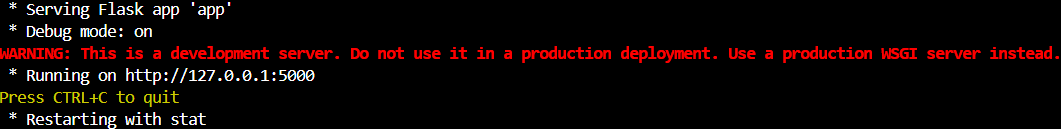
def get\_users():

return jsonify(users)

if name == ' main ':

app.run(debug=True)

**Output:**

****

navigate to <http://127.0.0.1:5000/users> in a web browser. You should see a static HTML page with the expected content

**A screenshot of a computer

Description automatically generated**

# **PRACTICAL – 2**

**AIM:** A program that creates a RESTful API that allows users to create, read, update, and delete resource

**Procedure:**

Step 1: Write the Flask App Open app.py in a text editor and write the Flask application code:

**app.py:**

from flask import Flask, jsonify, request

app = Flask( name )

books = [

{'id': 1, 'title': 'Book 1', 'author': 'Author 1'},

{'id': 2, 'title': 'Book 2', 'author': 'Author 2'},

{'id': 3, 'title': 'Book 3', 'author': 'Author 3'}

]

@app.route('/books', methods=['GET'])

def get\_books():

return jsonify(books)

@app.route('/books/<int:book\_id>', methods=['GET'])

def get\_book(book\_id):

book = next((b for b in books if b['id'] == book\_id), None)

if book:

return jsonify(book)

else:

return jsonify({'error': 'Book not found'}), 404

@app.route('/books', methods=['POST'])

def create\_book():

data = request.get\_json()

new\_book = {

'id': len(books) + 1,

'title': data['title'],

'author': data['author']

}

books.append(new\_book)

return jsonify(new\_book), 201

@app.route('/books/<int:book\_id>', methods=['PUT'])

def update\_book(book\_id):

book = next((b for b in books if b['id'] == book\_id), None)

if book:

data = request.get\_json()

book['title'] = data['title']

book['author'] = data['author']

return jsonify(book)

else:

return jsonify({'error': 'Book not found'}), 404

@app.route('/books/<int:book\_id>', methods=['DELETE'])

def delete\_book(book\_id):

global books

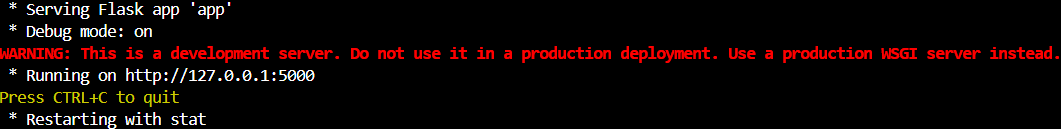
books = [b for b in books if b['id'] != book\_id]

return jsonify({'result': True})

if name == ' main ':

app.run(debug=True)

**Output:**

****

navigate to <http://127.0.0.1:5000/books> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated

# **PRACTICAL – 3**

**AIM:** A program that creates a RESTful API that authenticates users using a JSON Web Token.

**Procedure:**

Step 1: Write the Flask App Open app.py in a text editor and write the Flask application code:

**“pip install flask-jwt-extended”**

**app.py:**

from flask import Flask, jsonify, request

from flask\_jwt\_extended import JWTManager, jwt\_required,

create\_access\_token

app = Flask( name )

# Set up Flask-JWT-Extended

app.config['JWT\_SECRET\_KEY'] = 'your-secret-key' # Replace with your

secret key

jwt = JWTManager(app)

# Dummy user data (replace with a proper user database in a real

application)

users = {

'user1': {'password': 'password1'},

'user2': {'password': 'password2'}

}

# Route to generate a JWT token upon login

@app.route('/login', methods=['POST'])

def login():

data = request.get\_json()

username = data.get('username')

password = data.get('password')

if username in users and users[username]['password'] == password:

access\_token = create\_access\_token(identity=username)

return jsonify(access\_token=access\_token)

else:

return jsonify({'error': 'Invalid username or password'}), 401

# Protected route that requires a valid JWT token for access

@app.route('/protected', methods=['GET'])

@jwt\_required()

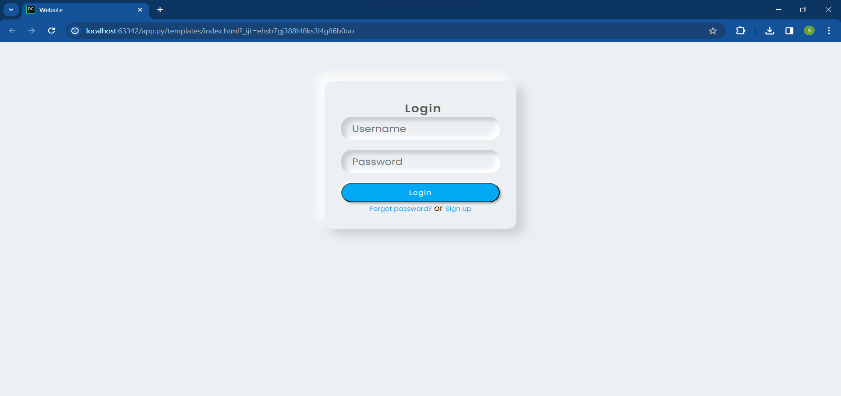
def protected():

current\_user = jwt.get\_jwt\_identity()

return jsonify(logged\_in\_as=current\_user), 200

if name == ' main ':

app.run(debug=True)

**Expected Output:**  

# **PRACTICAL – 4**

**AIM:** A program that creates a RESTful API that paginates the results of a query to improve performance.

**Procedure:**

Step 1: Write the Flask App Open app.py in a text editor and write the Flask application code:

**app.py :**

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

# Dummy data (replace with your actual data source)

items = [f'Item {i}' for i in range(1, 101)]

# Route that supports pagination

@app.route('/items', methods=['GET'])

def get\_items():

page = int(request.args.get('page', 1))

per\_page = int(request.args.get('per\_page', 10))

start\_index = (page - 1) \* per\_page

end\_index = start\_index + per\_page

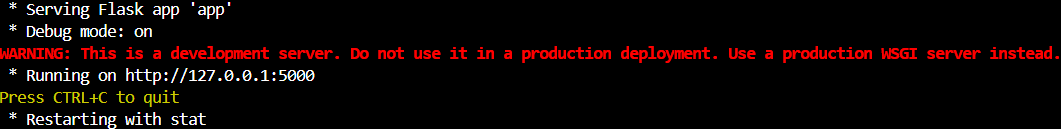
paginated\_items = items[start\_index:end\_index]

return jsonify({'items': paginated\_items, 'page': page, 'per\_page': per\_page, 'total\_items': len(items)})

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**Output:**

****

navigate to <http://127.0.0.1:5000/items> in a web browser. You should see a static HTML page with the expected content

**A screenshot of a computer

Description automatically generated**

# **PRACTICAL – 5**

**AIM:** A program that creates a RESTful API that supports data validation and error handling.

**Procedure:**

Step 1: Write the Flask App Open app.py in a text editor and write the Flask application code:

**app.py :**

from flask import Flask

from flask\_restful import Resource, Api, reqparse

app = Flask(\_\_name\_\_)

api = Api(app)

# Dummy data (replace with your actual data source)

items = {'1': {'name': 'Item 1', 'price': 10.99},

'2': {'name': 'Item 2', 'price': 19.99}}

# Request parser for input validation

parser = reqparse.RequestParser()

parser.add\_argument('name', type=str, required=True, help='Name cannot be blank')

parser.add\_argument('price', type=float, required=True, help='Price cannot be blank')

class ItemResource(Resource):

def get(self, item\_id):

item = items.get(item\_id)

if item:

return item

else:

return {'error': 'Item not found'}, 404

def put(self, item\_id):

args = parser.parse\_args()

items[item\_id] = {'name': args['name'], 'price': args['price']}

return items[item\_id], 201

def delete(self, item\_id):

if item\_id in items:

del items[item\_id]

return {'result': True}

else:

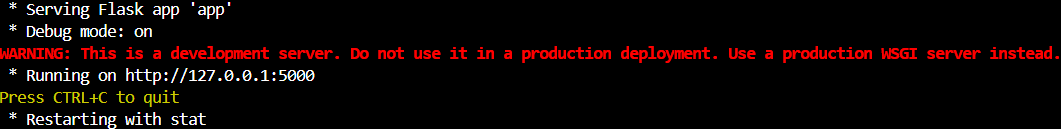
return {'error': 'Item not found'}, 404

api.add\_resource(ItemResource, '/items/<item\_id>')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Output:**

****

navigate to <http://127.0.0.1:5000/items/1> in a web browser. You should see a static HTML page with the expected content

A screenshot of a computer

Description automatically generated