## Practical 1: Define a simple services like Converting Rs into Dollar and Call it from different platform like JAVA and .NET

```# flask\_currency\_converter.py

from flask import Flask, request, jsonify

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

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

def convert\_currency():

try:

data = request.json

amount\_in\_rs = data['amount\_in\_rs']

conversion\_rate = 0.012 # Replace with the actual conversion rate

amount\_in\_usd = amount\_in\_rs \* conversion\_rate

return jsonify({'amount\_in\_usd': format(amount\_in\_usd, '.4f')})

except Exception as e:

return jsonify({'error': str(e)}), 500

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

app.run(debug=True)

```

```// Java code to call the service

import java.net.HttpURLConnection;

import java.net.URL;

import java.io.OutputStream;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.Scanner;

public class JavaClient {

public static void main(String[] args) throws Exception {

String url = "http://localhost:5000/convert";

// JSON payload

System.out.print("Enter the amount you want to convert: ");

Scanner scanner = new Scanner(System.in);

double amountInRs = scanner.nextDouble();

String jsonInputString = "{\"amount\_in\_rs\": "+amountInRs+"}";

scanner.close();

URL obj = new URL(url);

HttpURLConnection con = (HttpURLConnection) obj.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("Content-Type", "application/json");

// Send POST request

con.setDoOutput(true);

OutputStream os = con.getOutputStream();

os.write(jsonInputString.getBytes());

os.flush();

os.close();

// Get response

int responseCode = con.getResponseCode();

BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));

String inputLine;

StringBuffer response = new StringBuffer();

while ((inputLine = in.readLine()) != null) {

response.append(inputLine);

}

in.close();

// Print result

System.out.println("Response Code: " + responseCode);

System.out.println("Response: " + response.toString());

}

} ```

## Practical 2: Create a Simple SOAP service.

### Part 1: FahrenheitToCelsius

```import requests

import xml.etree.ElementTree as ET

temp = float(input("Enter a Fahrenheit value To convert in Celsius: "))

url = "https://www.w3schools.com/xml/tempconvert.asmx"

SOAPEnvelope = f"""<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<FahrenheitToCelsius xmlns="https://www.w3schools.com/xml/">

<Fahrenheit>{temp}</Fahrenheit>

</FahrenheitToCelsius >

</soap:Body>

</soap:Envelope>"""

headers = {

"Content-Type": "text/xml; charset=utf-8",

"SOAPAction": "https://www.w3schools.com/xml/FahrenheitToCelsius" # Adding SOAPAction header

}

response = requests.post(url, data=SOAPEnvelope, headers=headers)

root =ET.fromstring(response.text)

for child in root.iter("{https://www.w3schools.com/xml/}FahrenheitToCelsiusResult"):

C2F=child.text

print(C2F) ```

### Part 2: CelsiusToFahrenheit

``` import requests

import xml.etree.ElementTree as ET

temp = float(input("Enter a Celsius value To convert in Fahrenheit: "))

url = "https://www.w3schools.com/xml/tempconvert.asmx"

SOAPEnvelope = f"""<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<CelsiusToFahrenheit xmlns="https://www.w3schools.com/xml/">

<Celsius>{temp}</Celsius>

</CelsiusToFahrenheit>

</soap:Body>

</soap:Envelope>"""

headers = {

"Content-Type": "text/xml; charset=utf-8",

"SOAPAction": "https://www.w3schools.com/xml/CelsiusToFahrenheit" # Adding SOAPAction header }

response = requests.post(url, data=SOAPEnvelope, headers=headers)

root =ET.fromstring(response.text)

for child in root.iter("{https://www.w3schools.com/xml/}CelsiusToFahrenheitResult"):

C2F=child.text

print(C2F) ```

## **Practical 3: Create a Simple REST Service.**

### Part 1: ‘simple GET request’

```from flask import Flask

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

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

def welcome():

return "welcome to pyhton web service"

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

app.run() ```

### Part 2: ‘Dealing with books record’

```from flask import Flask, Response, jsonify, request

import xml.etree.ElementTree as Et

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

#Sample Data

books = [

{

'id':1,

'title':'Book 1',

'author':'Author Name',

},

{

'id':2,

'title':'Book 2',

'author':'Author 2'

}

]

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

def get\_all\_books():

root = Et.Element('books')

for book in books:

xml\_book = Et.SubElement(root, 'book')

xml\_book.set('id', str(book['id']))

title = Et.SubElement(xml\_book, 'title')

title.text = book['title']

author = Et.SubElement(xml\_book, 'author')

author.text = book['author']

xml\_string = Et.tostring(root, encoding='utf-8')

return Response(xml\_string, mimetype="text/xml")

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

def get\_one\_book(book\_id):

book = next((b for b in books if b['id']==book\_id))

if book:

return jsonify(book)

else:

return jsonify({"Message": "Book not Found"},404)

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

def add\_a\_book():

new\_book = {

'id':len(books)+1,

'title': request.json['title'],

'author': request.json['author'],

}

books.append(new\_book)

print(books)

return jsonify(new\_book)

#update book detail

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

def update\_book(book\_id):

request\_data = request.get\_json()

book = next((b for b in books if b['id']==book\_id))

if book:

book.update(request\_data)

return jsonify(book)

else:

return jsonify({"Message":"No Book Found"}), 404

#delete a book info from its id

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

def remove\_book(book\_id):

book = next((bi for bi in books if bi['id'] == book\_id))

if book is None:

return jsonify({"error":"Book Not Found!"})

books.remove(book)

print(books)

return jsonify({"message":"Successfully removed the book."})

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

app.run() ```

## Practical 4: Develop application to consume Google’s search / Google’s Map RESTful Web.

## Practical 5: Installation and Configuration of virtualization using KVM.

## Practical 6: Develop application to download image / video from server or upload image / video to server using MTOM techniques.

**``` #web\_server.py**

**from** flask import Flask, request, send\_file

import os

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

UPLOAD\_FOLDER = 'uploads'

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

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

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

def upload\_file():

try:

file = request.files['file']

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

file.save(file\_path)

return 'File uploaded successfully'

except Exception as e:

return str(e), 500

@app.route('/download/<filename>', methods=['GET'])

def download\_file(filename):

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

if os.path.exists(file\_path):

return send\_file(file\_path, as\_attachment=True)

else:

return 'File not found', 404

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

app.run(debug=True)

**```**

**```#soap\_client.py**

import requests, os

server\_url = 'http://localhost:5000'

def download\_file(filename, download\_folder=''):

download\_url = f'{server\_url}/download/{filename}'

response = requests.get(download\_url)

if response.status\_code == 200:

file\_path = os.path.join(download\_folder, filename)

with open(file\_path, 'wb') as f:

f.write(response.content)

print(f"File '{filename}' downloaded successfully to '{download\_folder}'")

else:

print(f"Failed to download file '{filename}': {response.text}")

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

file\_to\_download = 'infinity.mp4'

download\_folder = 'resources'

download\_file(file\_to\_download**, download\_folder)**

**```**

## Practical 7: Implement FOSS-Cloud Functionality VSI (Virtual server Infrastructure) Infrastructure as a Service (IaaS), Storage.

``` from flask import Flask, request, jsonify

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

virtual\_servers = []

storage = {"used": 0, "total": 1000}

@app.route('/create\_server', methods=['POST'])

def create\_server():

global storage

try:

data = request.json

server\_name = data.get('server\_name')

cpu = data.get('cpu')

ram = data.get('ram')

if not all([server\_name, cpu, ram]):

raise ValueError("Invalid data. Server name, CPU, and RAM are required.")

cpu = int(cpu)

ram = int(ram)

if cpu <= 0 or ram <= 0:

raise ValueError("CPU and RAM must be positive integers.")

virtual\_server = {"name": server\_name, "cpu": cpu, "ram": ram}

virtual\_servers.append(virtual\_server)

storage["used"] += cpu \* ram

return jsonify({"message": "Server created successfully", "server": virtual\_server})

except Exception as e:

return jsonify({"error": str(e)}), 400

@app.route('/list\_servers', methods=['GET'])

def list\_servers():

return jsonify({"servers": virtual\_servers})

@app.route('/get\_storage\_status', methods=['GET'])

def get\_storage\_status():

return jsonify(storage)

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

app.run(debug=True)

```

## Practical 8: Implement FOSS Cloud Functionality- VSI Platform as a Service (PaaS).

``` from flaskimport Flask, request, jsonify

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

platform\_applications = []

@app.route('/create\_application', methods=['POST'])

def create\_application():

try:

data = request.json

app\_name = data.get('app\_name')

app\_type = data.get('app\_type')

if not all([app\_name, app\_type]):

raise ValueError("Invalid data. App name and app type are required.")

new\_application = {"app\_name": app\_name, "app\_type": app\_type, "provider":"Dee-coding"}

platform\_applications.append(new\_application)

return jsonify({"message": "Application created successfully", "application": new\_application})

except Exception as e:

return jsonify({"error": str(e)}), 400

@app.route('/list\_applications', methods=['GET'])

def list\_applications():

return jsonify({"applications": platform\_applications})

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

app.run(debug=True)

```

## Practical 9: Using AWS Flow Framework develop application that includes a simple workflow. Workflow calls an activity to print hello world to the console. It must define the basic usage of AWS Flow Framework, including defining contracts, implementation of activities and workflow coordination logic and worker programs to host them.

## Practical 10: Implementation of OpenStack with user and private network creation.