Practical 1

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

Software Tools Required:

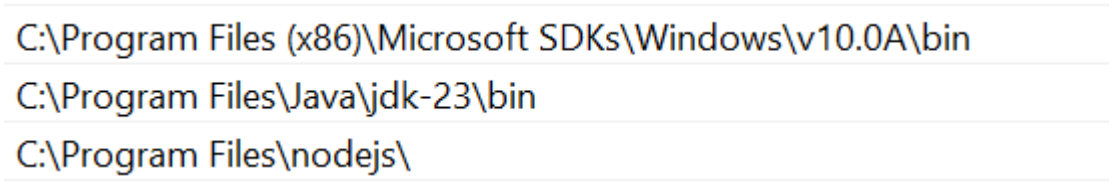
- **Code Editor**: Visual Studio Code (VS Code)
- **API Testing Software Application**: Postman
- **Framework**: Express.js (web framework)
- **Runtime Environment**: Node.js
- **Programming Languages**: JavaScript (for Node.js and Express.js), Java, C# (for .NET)

Downloads Required:

- Node.js: Node.js — Download Node.js®
- JDK: Java Downloads | Oracle India (x64 MSI Installer)
- Dotnet SDK: Download .NET 9.0 SDK (v9.0.101) - Windows x64 Installer
- Postman: Download Postman | Get Started for Free
  (Create Account/ Log in with Google)

After Downloading Node.js, JDK and Dotnet SDK, Set the Path in Environment Variable in User Variables > Path > Edit > New and Paste the Path for Node.js, JDK and Dotnet SDK

Demonstration:



Click OK to remaining opened Edit environment variable, Environment Variables and System Properties windows.

Now check the versions that shows path is set for Nodejs, Java JDK and Dotnet SDK

Here all versions are shown so path is set by checking versions for Nodejs, Java JDK and Dotnet SDK.

1. **Design the Currency Conversion Service:**

Create a simple API that takes an amount in Indian Rupees and returns the equivalent amount in US Dollars. You can use a simple formula for conversion, or you can fetch real-time exchange rates from a reliable source.

Example API endpoint:

POST/convert

Request:

"amount in rs”: 1000

Response:

{

"amount in usd": 14.5

}

2. **Implement the Service:**

You can implement the service using any programming language and framework. For simplicity, let's use Node.js with Express in this example,

i. **Set up a Node.js Project**

Create a Project Directory

Example: Directory Path: Documents in C Drive

Create a Folder Name: Cloud_Computing_Pratical1

C: \Documents\Cloud_Computing_Practical1

Select the Directory Path, Cut (Backspace) and Type cmd and hit enter

We will get inside the path now type the following commands

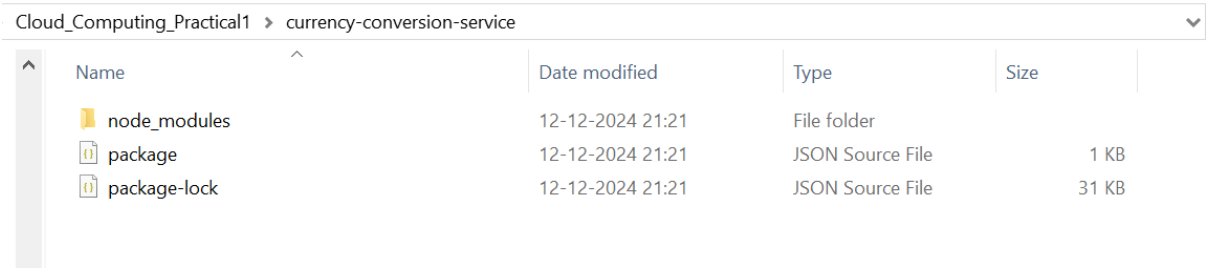
- mkdir currency-conversion-service : mkdir - Make directory
- cd currency-conversion-service : cd – change directory

- npm init -y : npm – node package manager initializes yes
- npm install express body-parser : To create simple server & handle incoming requests

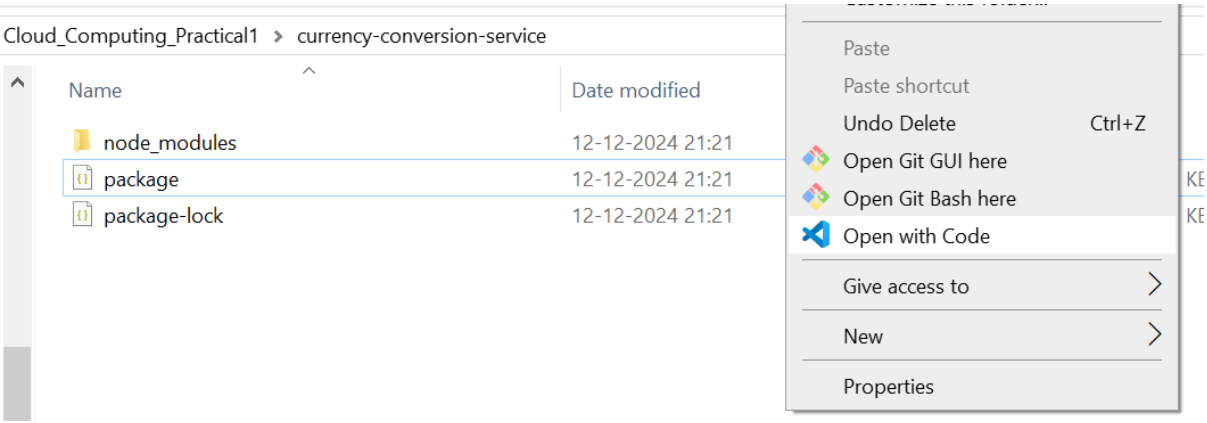
Demonstration:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Kishore\Documents\Cloud_Computing_Practical1>mkdir currency-conversion-service

C:\Users\Kishore\Documents\Cloud_Computing_Practical1>cd currency-conversion-service

C:\Users\Kishore\Documents\Cloud_Computing_Practical1\currency-conversion-service>npm init -y
Wrote to C:\Users\Kishore\Documents\Cloud_Computing_Practical1\currency-conversion-service\package.json:

{
  "name": "currency-conversion-service",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



C:\Users\Kishore\Documents\Cloud_Computing_Practical1\currency-conversion-service>npm install express body-parser

added 72 packages, and audited 73 packages in 3s

17 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Kishore\Documents\Cloud_Computing_Practical1\currency-conversion-service>
```

Inside currency-conversion-service > this is the folder architecture after initializing npm and express body-parser.



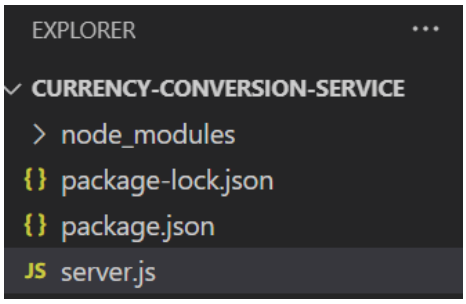
Now Open with VS Code



JavaScript (for Node.js and Express.js)

File Architecture:

Create a file to implement the service with JavaScript Programming language



Filename: server.js

Code:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000;
app.use(bodyParser.json());
app.post('/convert', (req, res) => {
    const amountInRs = req.body.amount_in_rs;
    // Perform the conversion (use a real exchange rate or a fixed rate for simplicity)
    const conversionRate = 0.0145;
    const amountInUsd = amountInRs * conversionRate;
    res.json({ amount_in_usd: amountInUsd });
});
app.listen(port, () => {
    console.log(`Currency conversion service running on http://localhost:${port}`);
```

```
});
```

Output:

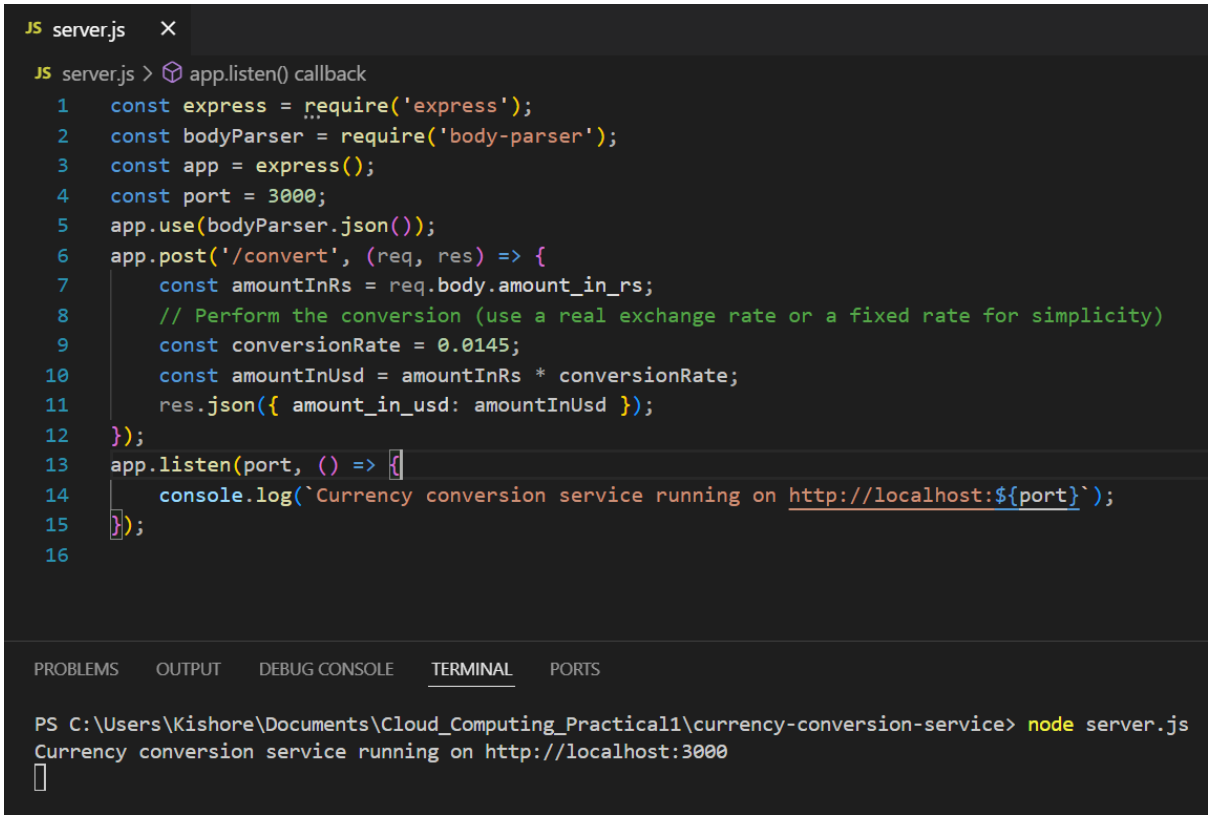
Click View > Terminal:

PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\currency-conversion-service> node server.js

Currency conversion service running on http://localhost:3000

The Output is successful indicating the service is running on http://localhost:3000

Demonstration:



Postman

API Testing Software Application to Test the API

Open Postman

Send a POST Request:

URL: http://localhost:3000/convert

Body: Set the body to raw JSON and include:

{

"amount_in _rs": 1000

}

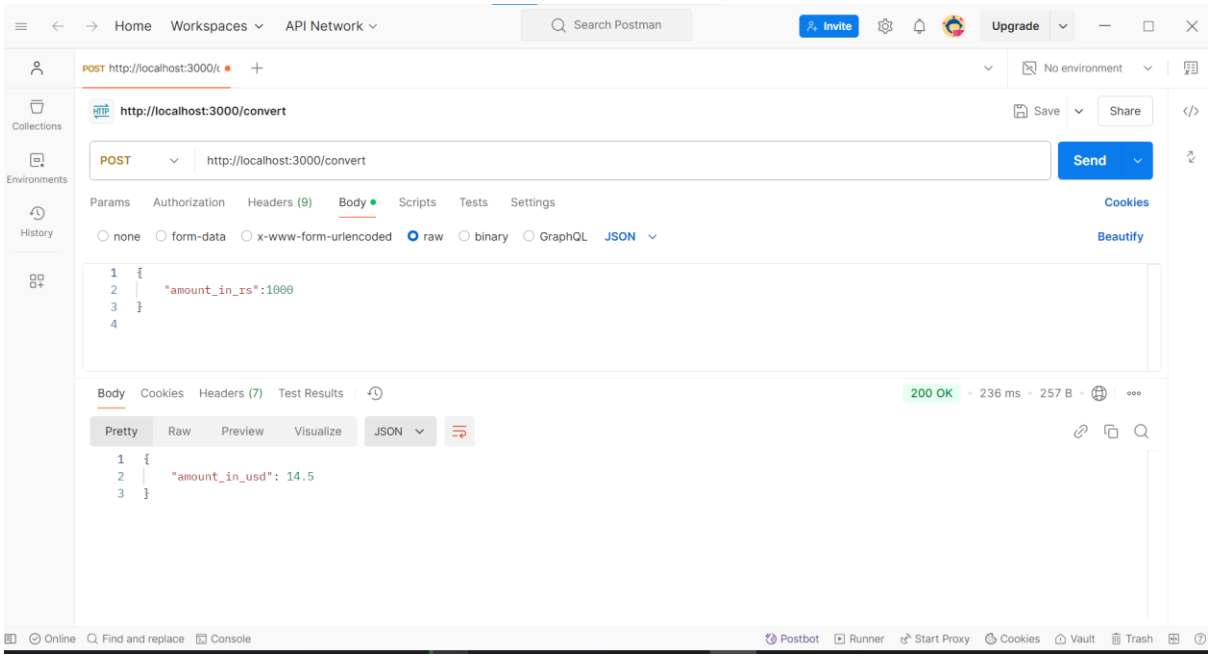
Response:

{

"amount_in _usd": 14.5

}

Demonstration:



Successfully output retrieved in Postman to test API.

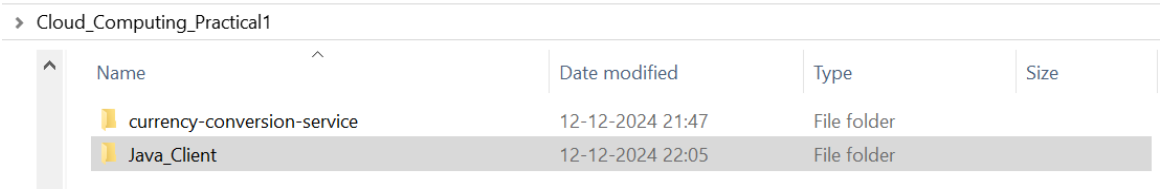
Now, to call from 2 different platforms Java and .NET Programming Language.

Java Programming Language

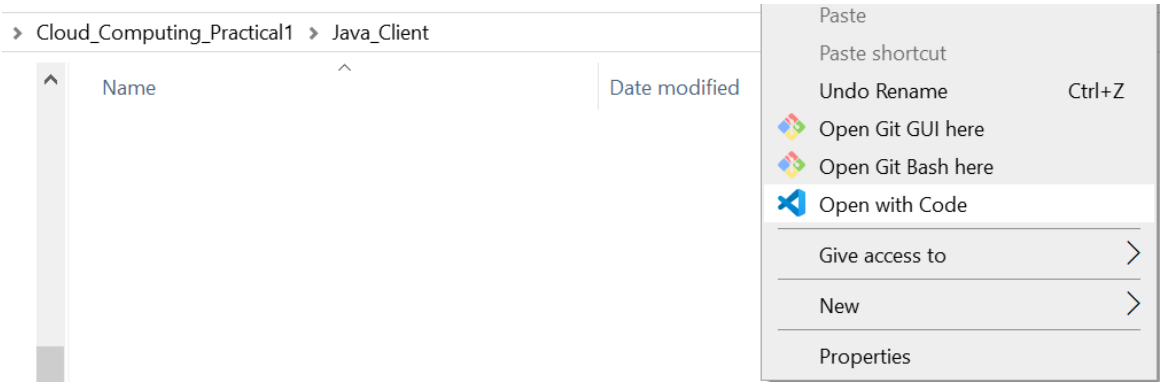
1. Java Client: Implement a Java Client to call the currency conversion service. You can use libraries like HttpClient for making HTTP requests.

Create folder name: Java_Client in Cloud_Computing_Practical1

Demonstration:

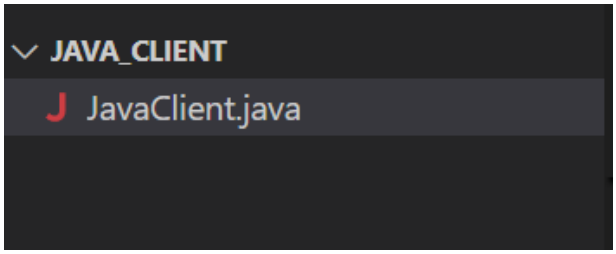


Open Java_Client and Open with VS Code



Extension: Install Java in VS Code

File Architecture:



Filename: JavaClient.java

Code:

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpRequest.BodyPublishers;
import java.net.http.HttpResponse.BodyHandlers;


public class JavaClient {
    public static void main(String[] args) {
        HttpClient client = HttpClient.newHttpClient();
        String uri = "http://localhost:3000/convert";
        String requestBody = "{ \"amount_in_rs\": 1000 }";

        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(uri))
                .header("Content-Type", "application/json")
                .POST(BodyPublishers.ofString(requestBody))
                .build();

        client.sendAsync(request, BodyHandlers.ofString())
                .thenApply(HttpResponse::body)
                .thenAccept(System.out::println)
                .join();
    }
}
```

Output:

Click View > Terminal:

```
PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\Java_Client> javac JavaClient.java
PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\Java_Client> java JavaClient
{"amount_in_usd":14.5}
```

The Output is Successful that we called the server.js with making HTTP request

Demonstration:

JavaClient.java

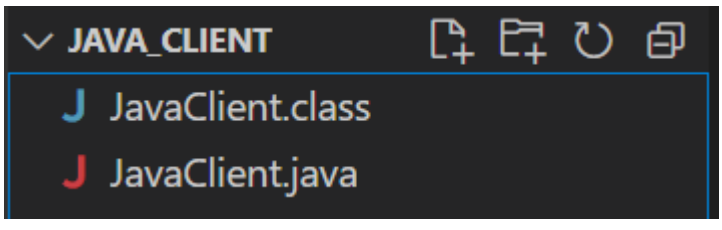
```java
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpRequest.BodyPublishers;
import java.net.http.HttpResponse.BodyHandlers;

public class JavaClient {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        HttpClient client = HttpClient.newHttpClient();
        String uri = "http://localhost:3000/convert";
        String requestBody = "{ \"amount_in_rs\": 1000 }";

        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(uri))
                .header(name:"Content-Type", value:"application/json")
                .POST(BodyPublishers.ofString(requestBody))
                .build();

        client.sendAsync(request, BodyHandlers.ofString())
                .thenApply(HttpResponse::body)
                .thenAccept(System.out::println)
                .join();
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\Java_Client> javac JavaClient.java
PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\Java_Client> java JavaClient
{"amount_in_usd":14.5}
```

The java command is used to run the compiled Java bytecode. It starts the JVM, loads the specified .class file, and then executes the main method of that class.

1. Ensure you have a .class file: This file is generated by compiling a .java file using the javac command.
2. Run the Java program: Use the java command followed by the name of the class (without the .class extension) to execute the program**.**

Here JavaClient.class file will be created after running the program

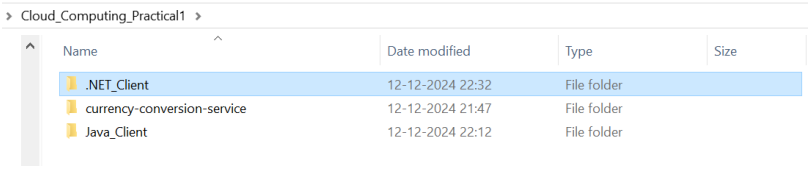


.NET Programming Language

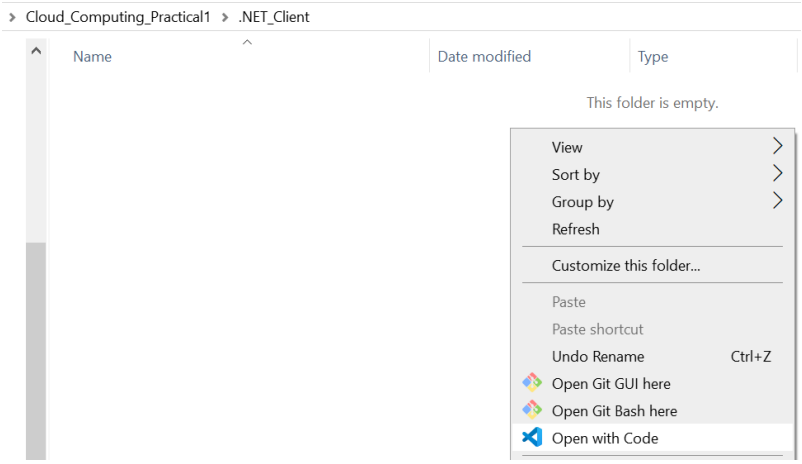
2. .NET Client: Implement a .NET Client to call the currency conversion service. You can use HttpClient in C#.

Create folder name: .NET_Client in Cloud_Computing_Practical1

Demonstration:

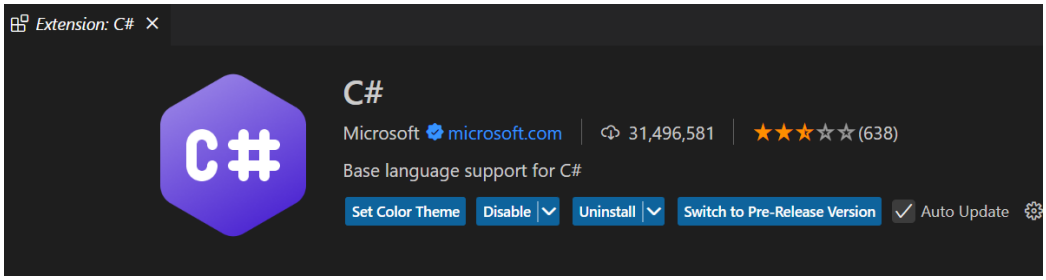


Open .NET_Client and Open with VS Code

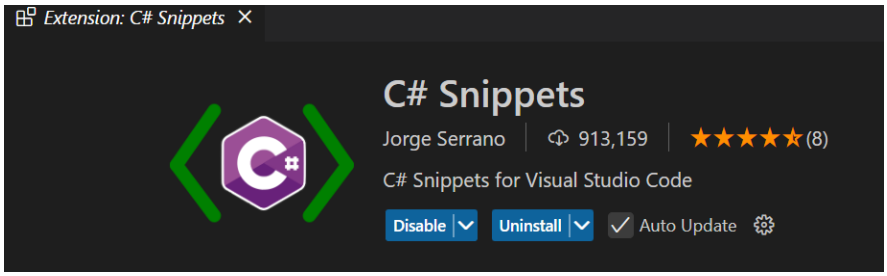


Extension: 4 Extensions needed to work in C#

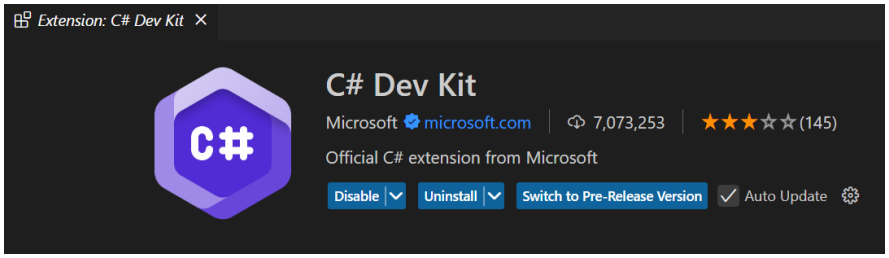
C#



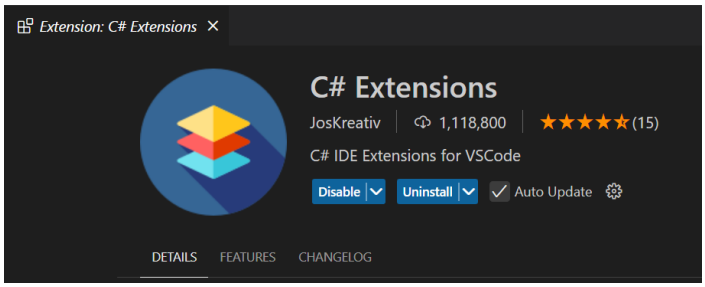
C# Snippet



C# Dev Kit



C# Extensions

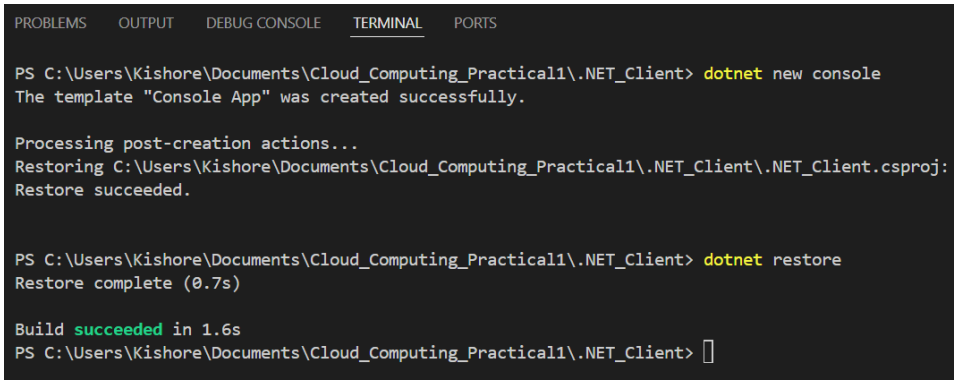


File Architecture:

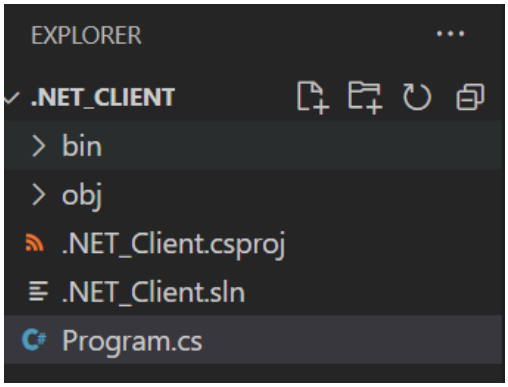
Click on View > Terminal and Type the following command

- dotnet new console
- dotnet restore

Demonstration:



After entering dotnet new console command the file architecture will generate the .NET Project with Program.cs file shown below.



Filename: Program.cs

Code:

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

class Program
{
   static async Task Main()
   {
      try
      {
         var client = new HttpClient();
         var apiUrl = "http://localhost:3000/convert";

         // Create the request body with correct JSON format
         var requestBody = new
         {
            amount_in_rs = 1000
         };
         var jsonRequestBody = System.Text.Json.JsonSerializer.Serialize(requestBody);

         // Send an HTTP POST request to the API with the request body
         var response = await client.PostAsync(apiUrl, new StringContent(jsonRequestBody,
Encoding.UTF8, "application/json"));

         // Check the response status code
         if (response.IsSuccessStatusCode)
         {
            Console.WriteLine("Response Code: " + response.StatusCode);
            Console.WriteLine("Response Body: " + await response.Content.ReadAsStringAsync());
         }
         else
         {
            Console.WriteLine("Error: " + response.StatusCode);
            Console.WriteLine("Details: " + await response.Content.ReadAsStringAsync());
         }
      }
      catch (HttpRequestException e)
      {
         Console.WriteLine("An error occurred: " + e.Message);
      }
   }
}
```

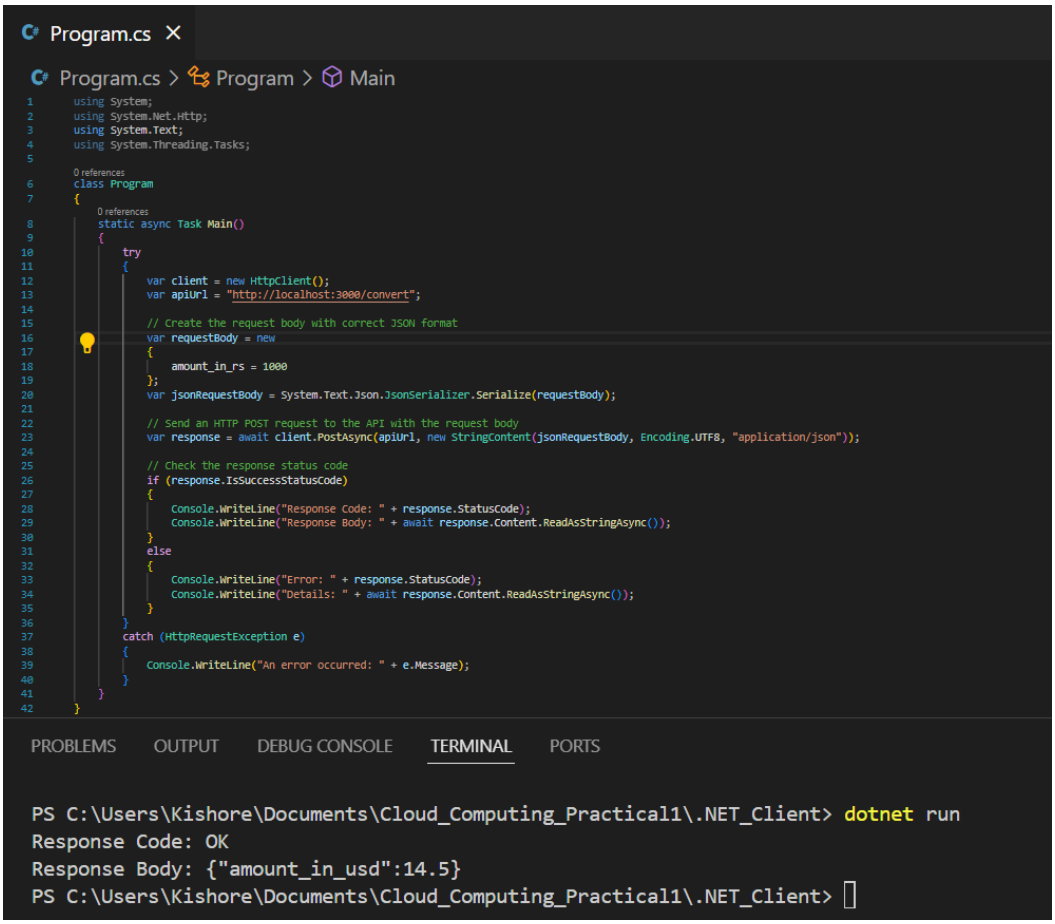
Output:

Click View > Terminal:

PS C:\Users\Kishore\Documents\Cloud_Computing_Practical1\.NET_Client> dotnet run
Response Code: OK
Response Body: {"amount_in_usd":14.5}

The Output is Successful that we called the server.js with making HTTP request

Demonstration:



Conclusion: The Practical worked successfully after creating currency conversion in postman with javascript and calling with Java and .NET by HTTP Requests.

Practical 2

Create a Simple SOAP service.

Software Tools Required:

- **Code Editor:** Eclipse IDE
- **Build Automation Tool:** Apache Maven
- **SOAP Testing Tool:** SOAPUI
- **Framework:** Apache CXF
- **Programming Language:** Java

Downloads Required:

- Apache Maven: Download Apache Maven – Maven
- JDK: Java Downloads | Oracle India (x64 MSI Installer)
- Eclipse IDE: Eclipse downloads - Select a mirror | The Eclipse Foundation
- SOAP Testing Tool: Download REST & SOAP Automated API Testing Tool | Open Source | SoapUI (SoapUI Open Source)

After Downloading JDK and Maven, Set the Path in Environment Variable in User Variables > Path > Edit > New and Paste the Path for JDK and Maven and check the versions for both in command prompt.

Demonstration:

Path Set

```
C:\Program Files\Java\jdk-23\bin
D:\apache-maven-3.9.9\bin
```

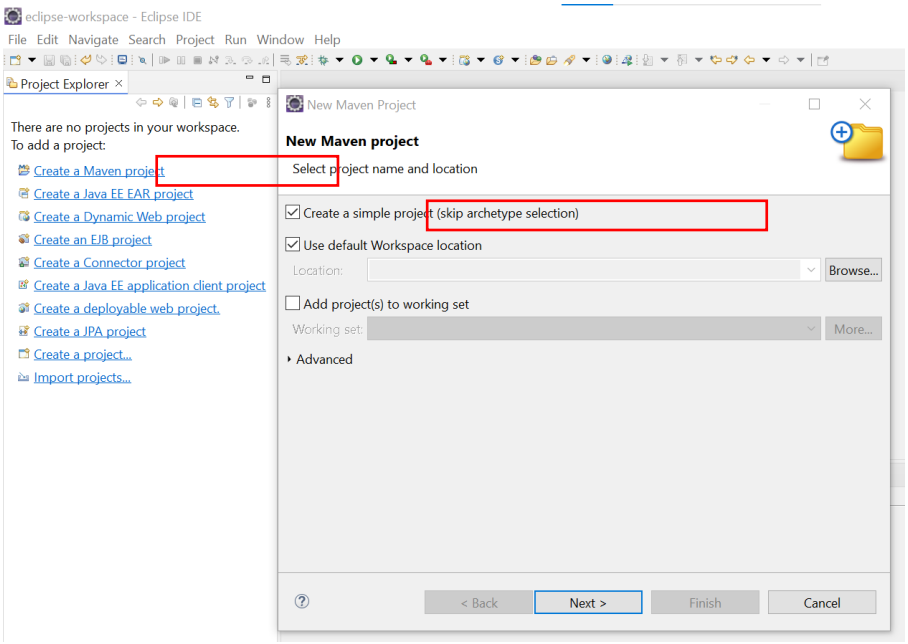
Version checked in command prompt

```
C:\Users\Kishore>mvn -v
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\apache-maven-3.9.9
Java version: 23.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-23
Default locale: en_IN, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Kishore>java --version
java 23.0.1 2024-10-15
Java(TM) SE Runtime Environment (build 23.0.1+11-39)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.1+11-39, mixed mode, sharing)
```

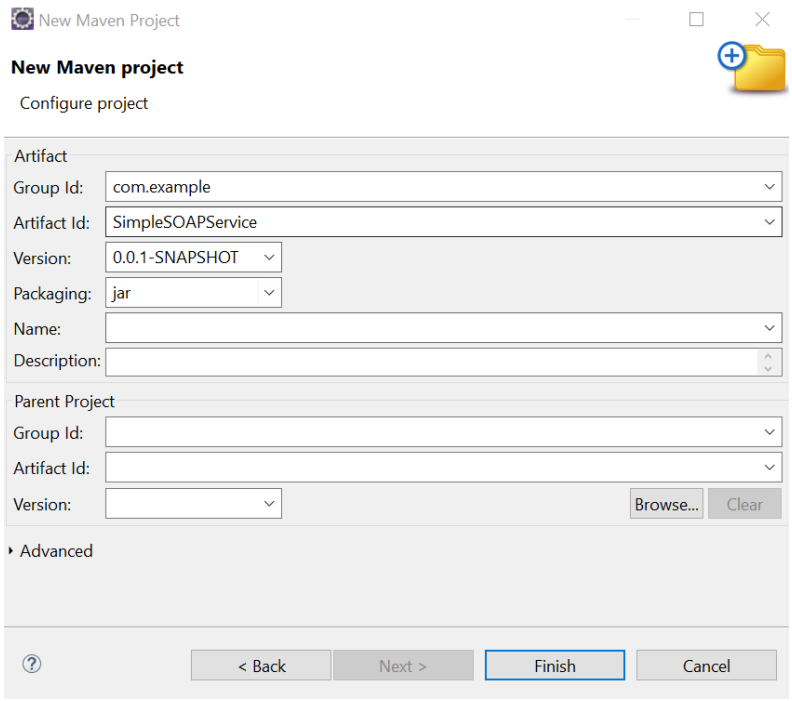
Java Programming Language

Project Name: SimpleSOAPService

Step 1: Open Eclipse IDE, Create a Maven Project, Check – Create a simple project (skip archetype selection) and Click Next.



Step 2: In New Maven Project window add the following details – Group id: com.example, Artifact Id: SimpleSOAPService and Version: 0.0.1- SNAPSHOT and click Finish



Step 3: After creating the Maven project, update the pom.xml to include the necessary dependencies for Apache CXF and JAX-WS. Here's the full pom.xml file.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>SimpleSOAPService</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
      <!-- Apache CXF for JAX-WS -->
      <dependency>
         <groupId>org.apache.cxf</groupId>
         <artifactId>cxf-rt-frontend-jaxws</artifactId>
         <version>3.4.5</version>
```

```
    </dependency>

    <!-- Apache CXF HTTP Transport -->
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-transports-http</artifactId>
      <version>3.4.5</version>
    </dependency>

    <!-- Apache CXF HTTP Jetty Transport (for embedded HTTP server) -->
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-transports-http-jetty</artifactId>
      <version>3.4.5</version>
    </dependency>

    <!-- Jakarta XML WS (JAX-WS API) -->
    <dependency>
      <groupId>jakarta.xml.ws</groupId>
      <artifactId>jakarta.xml.ws-api</artifactId>
      <version>2.3.3</version>
    </dependency>

    <!-- Java Annotations -->
    <dependency>
      <groupId>javax.annotation</groupId>
      <artifactId>javax.annotation-api</artifactId>
      <version>1.3.2</version>
    </dependency>
  </dependencies>
```

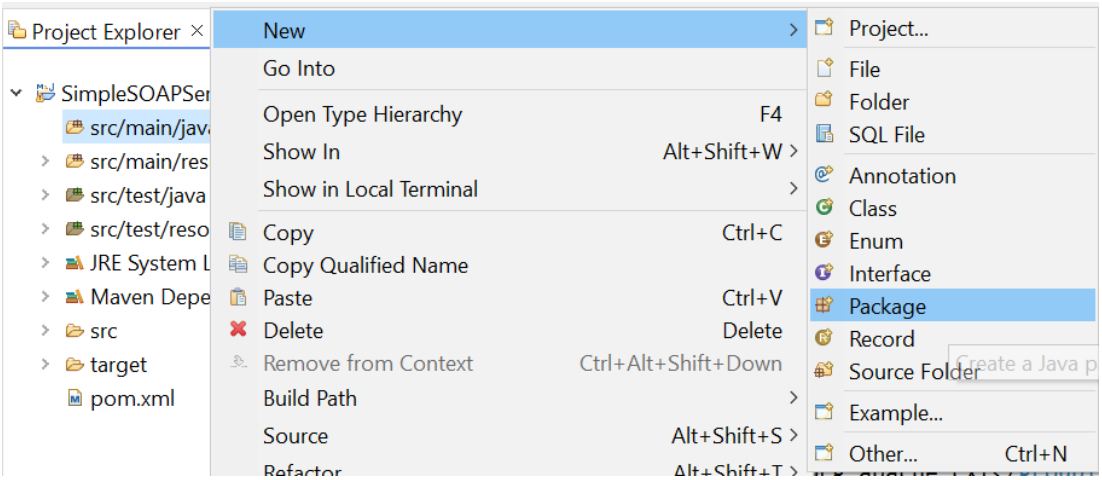
</project>

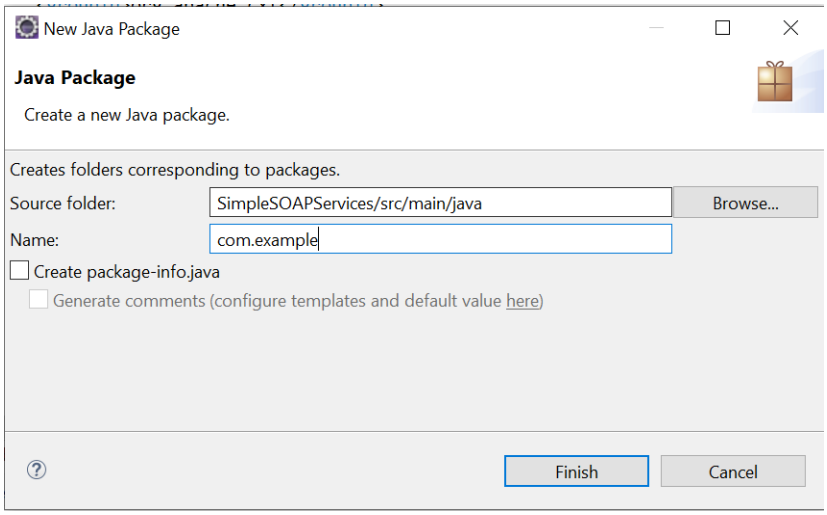
After adding dependencies Save it by Ctrl+S

Step 4: Create the Service Interface

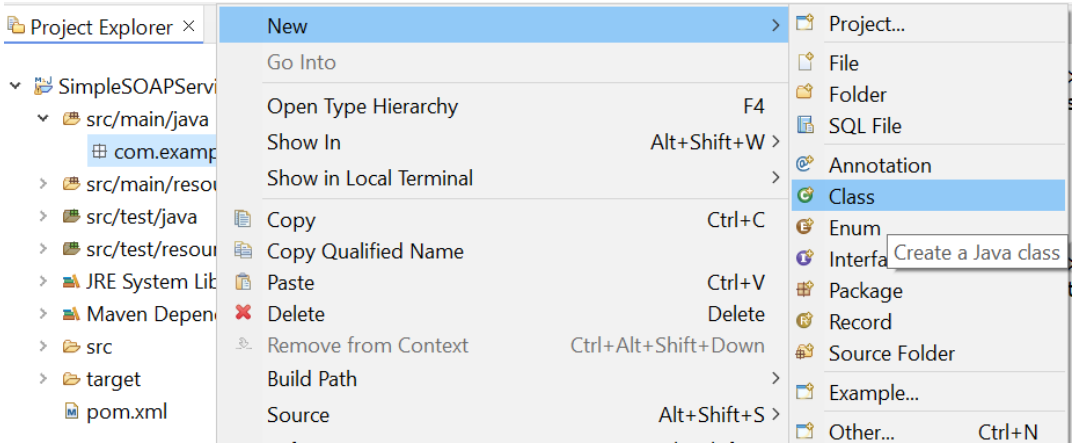
Now, create a simple SOAP service interface to define the operations.

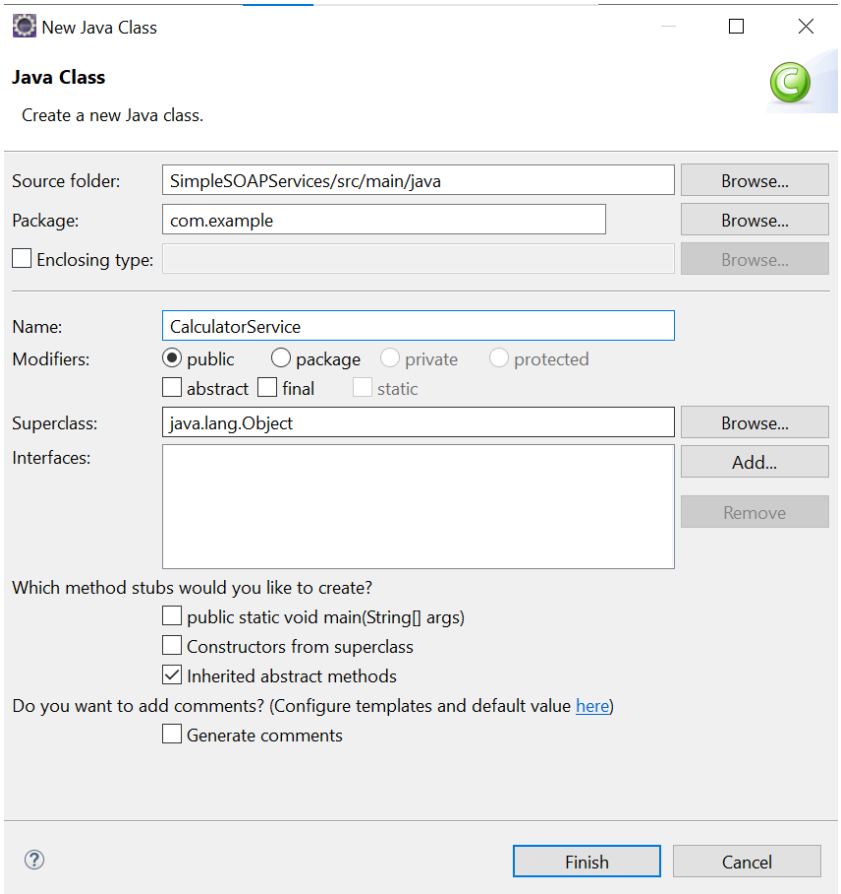
1. **Right-click** on src/main/java → **New** → **Package** and name it com.example





1. **Right-click** on the com.example package → **New** → **Class** and name it CalculatorService.java.





File Name: CalculatorService.java

Code:

```
package com.example;


import javax.jws.WebMethod;

import javax.jws.WebService;


@WebService

public interface CalculatorService {

        @WebMethod

        int add(int num1, int num2);


        @WebMethod

        int subtract(int num1, int num2);

}
```

Save it

Step 5: Implement the Service

Now, create a class that implements the CalculatorService interface.

1. **Right-click** on com.example → **New** → **Class** and name it CalculatorServiceImpl.java.
2. Add the following code to CalculatorServiceImpl.java:

File Name: CalculatorServiceImpl.java

Code:

```
package com.example;


import javax.jws.WebService;


@WebService(endpointInterface = "com.example.CalculatorService")
```

```
public class CalculatorServiceImpl implements CalculatorService {

    @Override
    public int add(int num1, int num2) {
        return num1 + num2;
    }
    @Override
    public int subtract(int num1, int num2) {
        return num1 - num2;
    }
}
```

Save It

Step 6: Create the SOAP Server

Next, create a class to publish the SOAP service.

1. **Right-click** on com.example → **New** → **Class** and name it SOAPServer.java.
2. Add the following code to SOAPServer.java:

File Name: SOAPServer.java

Code:

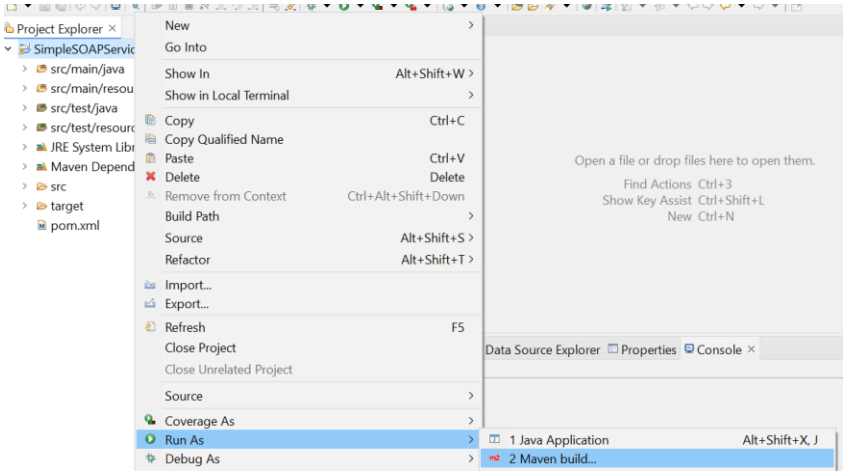
```
package com.example;

import javax.xml.ws.Endpoint;

public class SOAPServer {
    public static void main(String[] args) {
        // Create an instance of the service implementation
        CalculatorServiceImpl implementor = new CalculatorServiceImpl();

        // Publish the service
        String address = "http://localhost:8080/CalculatorService";
        Endpoint.publish(address, implementor);
```

```
        System.out.println("SOAP Service started at " + address);
    }
}
```

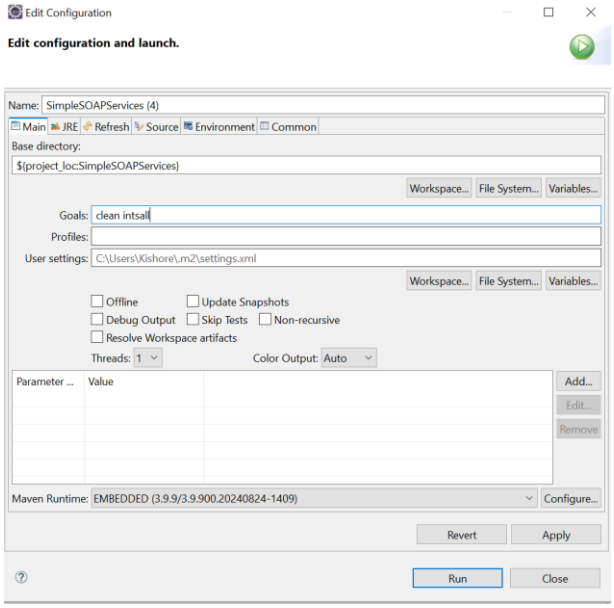
Save It

Output:

mvn clean install: The clean install command is commonly used in Maven to build and install a project. The purpose of running clean is to remove any previously compiled code or artifacts, ensuring that the next build starts fresh without using any old files that might cause conflicts or errors.



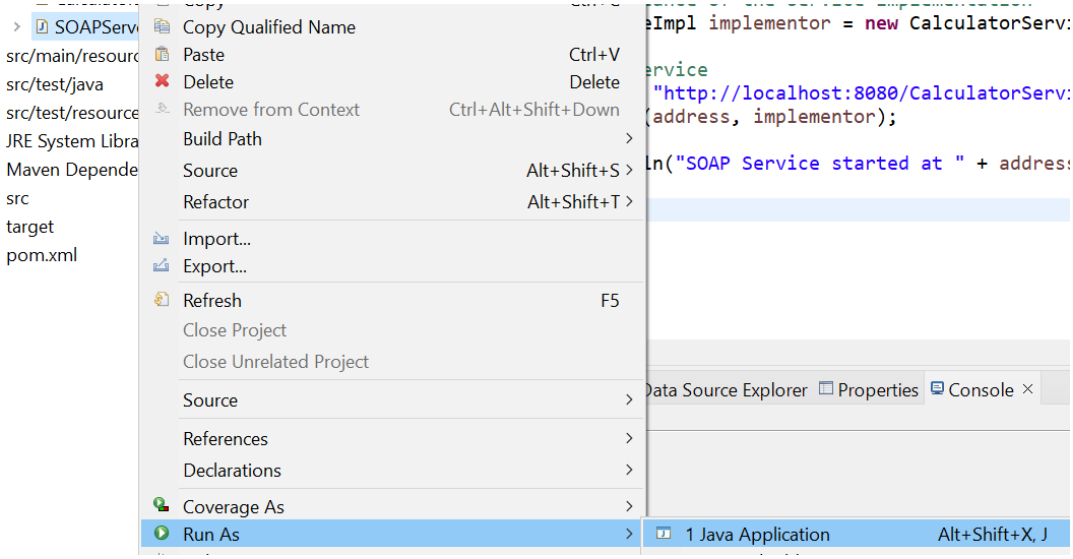
In Goal type clean install and run

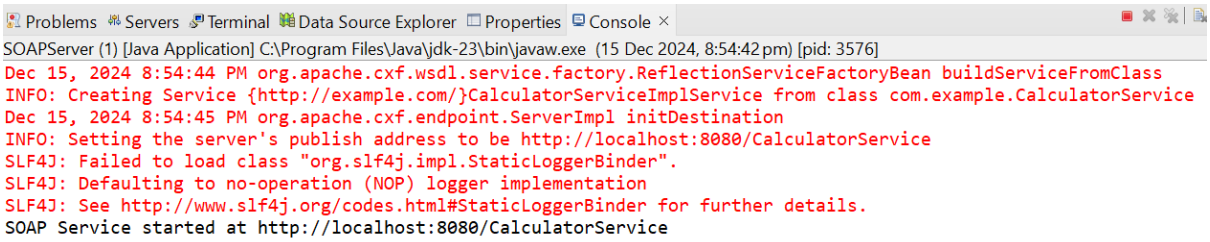


Run the Application

1. Right-click on SOAPServer.java → Run As → Java Application.
2. You should see the following output:

SOAP Service started at http://localhost:8080/CalculatorService

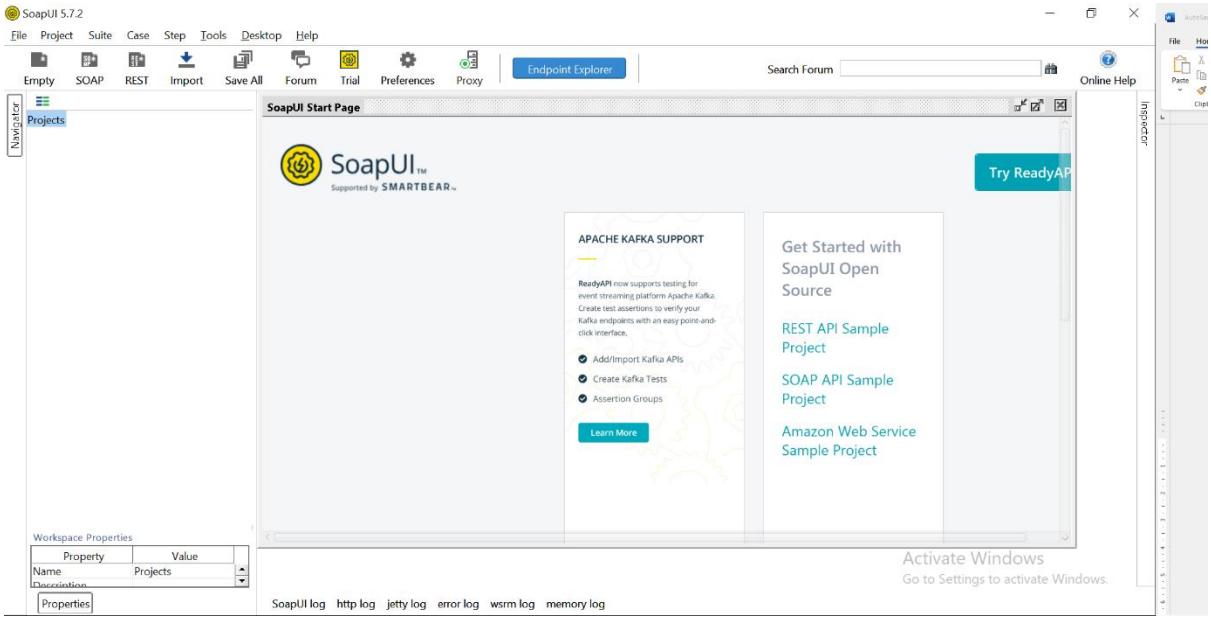


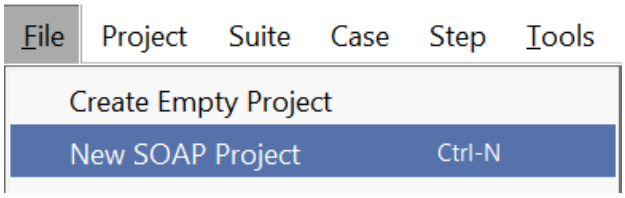


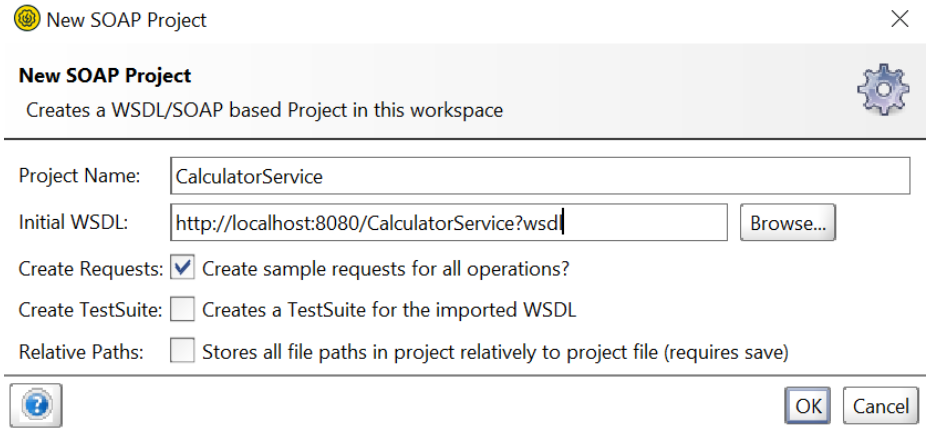
Step 7. Test the SOAP Service Using SOAPUI

1. Open SOAPUI and create a new project:
    1. Click File → New SOAP Project.
    2. In the Project Name field, enter CalculatorService.
    3. In the Initial WSDL field, enter:
    http://localhost:8080/CalculatorService?wsdl

SOAPUI:

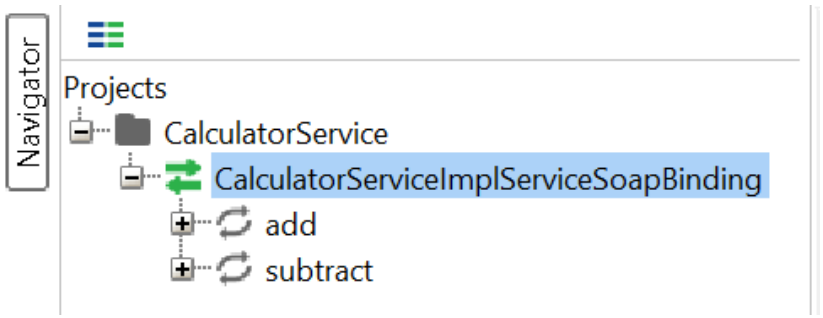


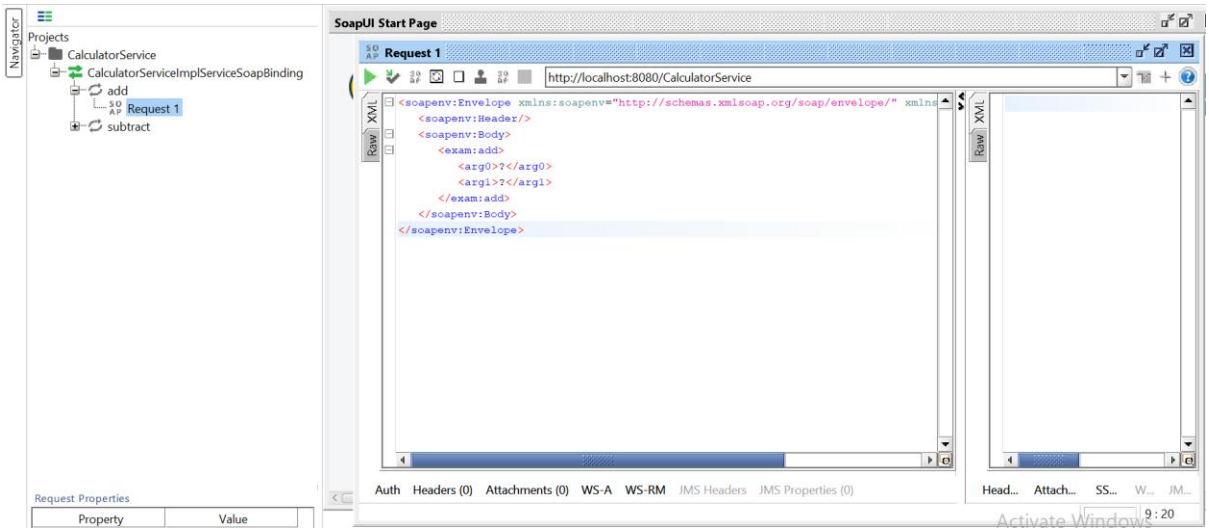




Click OK

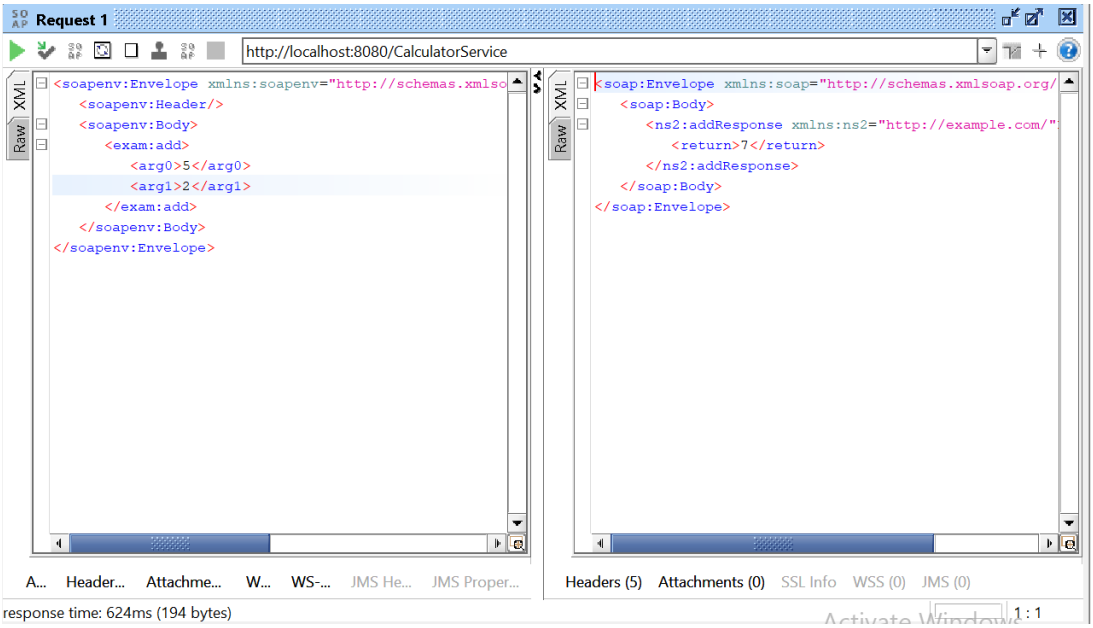
SOAPUI will import the WSDL and display the operations (add and subtract). You can now test the SOAP service by right-clicking on the operations and sending requests.





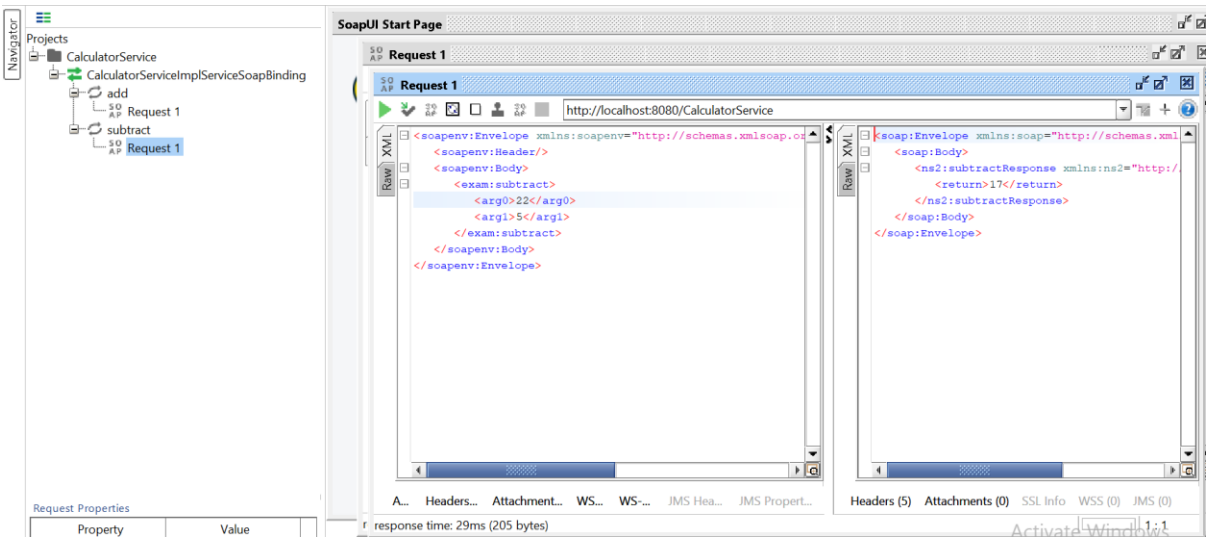
add > Request 1

Here, in add we got the request and edited 5 and 2 in argument tags and output displayed in xml.



subtract > Request 1

Here, in subtract we got the request and edited 22 and 5 in argument tags and output displayed in xml.



Conclusion: Why Use SoapUI to Create SOAP Services and Get Output in XML for Add and Subtract Operations

SoapUI is a popular tool for testing and interacting with SOAP (Simple Object Access Protocol) web services. It is used to create and test SOAP services, allowing developers and testers to ensure that their web services function correctly and handle various input scenarios.

Logic to Work with SOAP Services in SoapUI:

Creating SOAP Service:

- WSDL (Web Services Description Language): SoapUI uses the WSDL file to generate the request and response structure for SOAP services. The WSDL defines the operations, input/output messages, and service endpoint.
- SOAP Request and Response: In SoapUI, you can create SOAP requests based on the operations defined in the WSDL. For operations like "Add" and "Subtract," the SOAP request will include the parameters (e.g., two numbers) to be processed by the web service.

Practical 3

Create a Simple REST Service

Software Tools Required:

- **Code Editor**: Visual Studio Code (VS Code)
- **API Testing Software Application**: Postman
- **Framework**: Flask (micro web framework)

Downloads Required:

- Python: Download Python | Python.org
- Postman: Download Postman | Get Started for Free
  (Create Account/ Log in with Google)

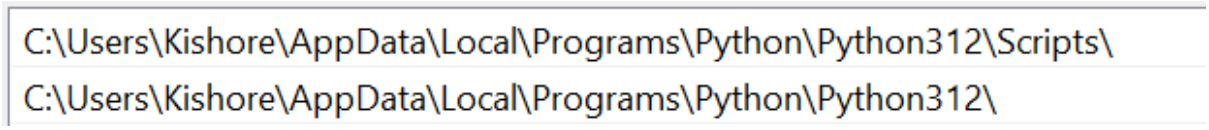
After Downloading Python, Set the Path in Environment Variable in User Variables > Path > Edit > New and Paste the Path for Python.

New version is 3.13, Here the version is 3.12 so following the same for new version.

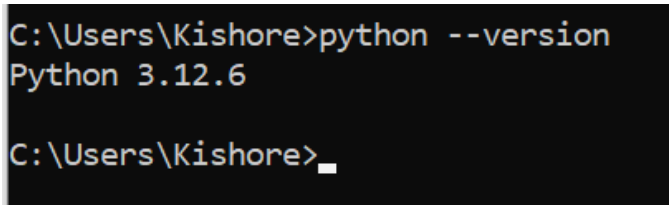
Till Python312: Copy Directory Path
Start > Python 3.13.1 (64 bit) > Open File Location > Python 3.13.1 (64 bit)

Till Python312\Scripts: Copy the Directory Path
Start > Python 3.13.1 (64 bit) > Open File Location > Python 3.13.1 (64 bit) > Open File Location > Scripts

Demonstration:



Check Version:



Step 1: Install Flask: Make sure you have Python installed on your system.
Open cmd and type the following command

pip install Flask

Demonstration:

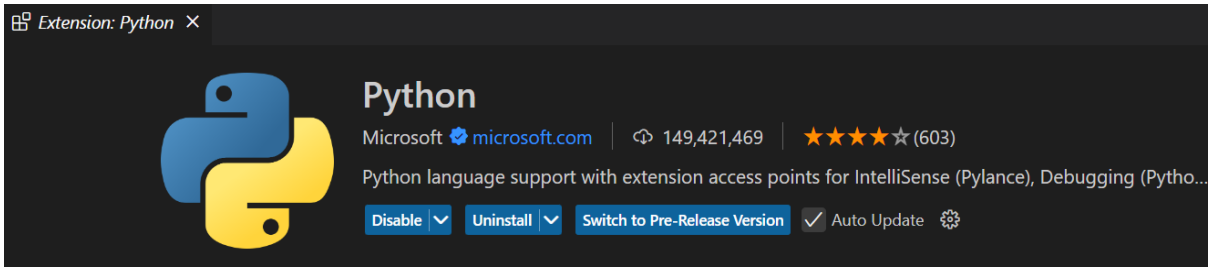
```
C:\Users\Kishore>pip install Flask
Requirement already satisfied: Flask in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Flask) (3.0.3)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Flask) (1.8.2)
Requirement already satisfied: colorama in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from click>=8.1.3->Flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\kishore\appdata\local\programs\python\python312\lib\site-packages (from Jinja2>=3.1.2->Flask) (2.1.5)

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Step 2: Create a folder name SimpleRESTService, Open with VS Code and create python file.

Check and Install Extensions for Python

Extensions:

Python



Python Debugger

Filename: app.py

Code:

```
from flask import Flask, jsonify, request
app = Flask(__name__)

# Sample data
data = [
    {'id': 1, 'name': 'Item 1'},
    {'id': 2, 'name': 'Item 2'},
]

# Endpoint to get all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify({'items': data})

# Endpoint to get a specific item by ID
@app.route('/items/<int:item_id>', methods=['GET'])
def get_item(item_id):
    item = next((item for item in data if item['id'] == item_id), None)
    if item:
        return jsonify({'item': item})
    else:
        return jsonify({'message': 'Item not found'}), 404
```

```
# Endpoint to add a new item
@app.route('/items', methods=['POST'])
def add_item():
    new_item = {'id': len(data) + 1, 'name': request.json['name']}
    data.append(new_item)
    return jsonify({'message': 'Item added successfully', 'item': new_item}), 201


# Run the application
if __name__ == '__main__':
    app.run(debug=True)
```

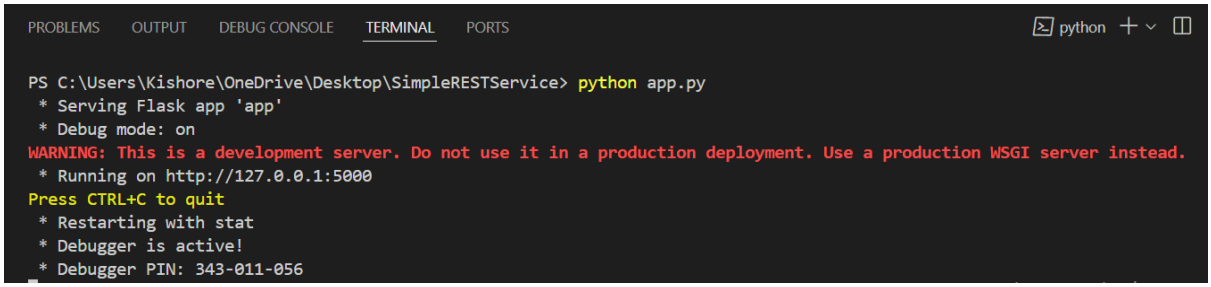
Output:

View>Terminal

Type

Path> **python app.py**

Demonstration:



Postman

API Testing Software Application to Test the API

Open Postman

 # Endpoint to get all items

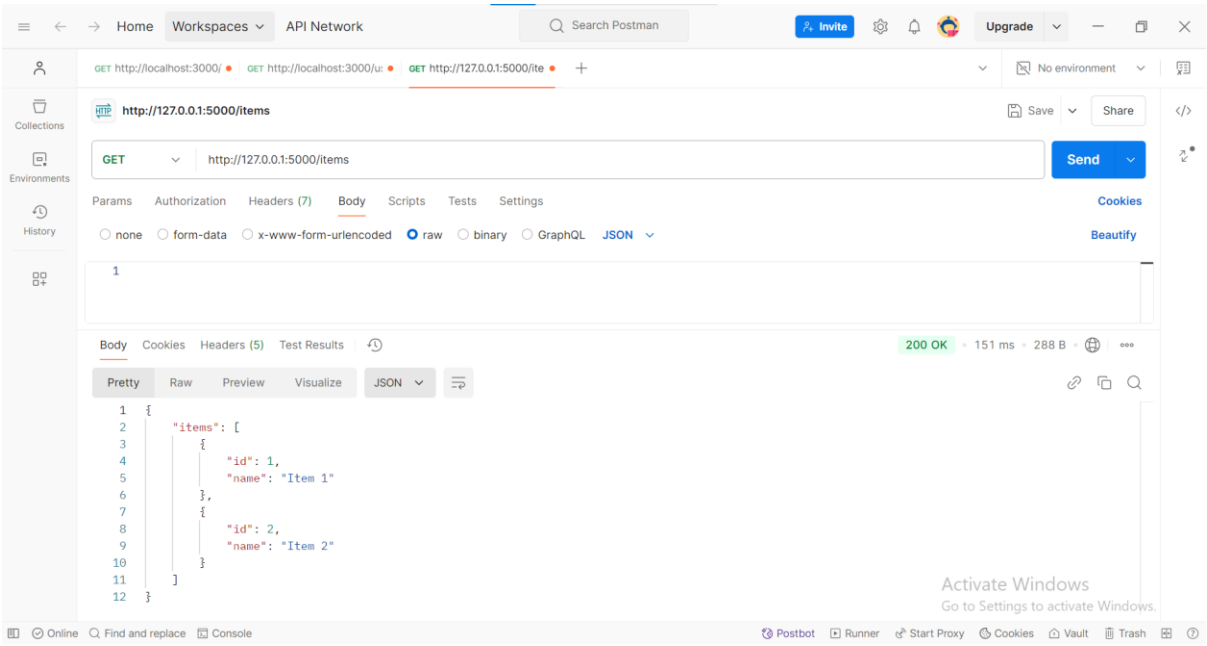
Send a GET Request:

URL: http://127.0.0.1:5000/items

Response:


```
{
  "items": [
    {
      "id": 1,
      "name": "Item 1"
    },
    {
      "id": 2,
      "name": "Item 2"
    }
  ]
}
```


Demonstration:



 # Endpoint to get a specific item by ID

Send a GET Request:

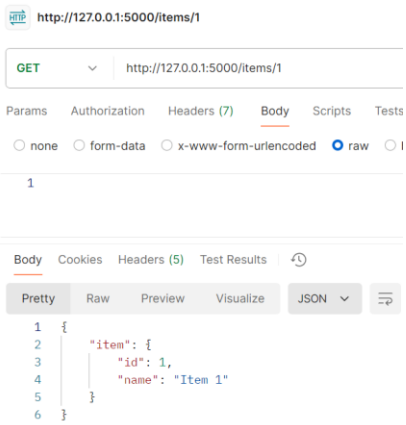
URL: http://127.0.0.1:5000/items/1

Response:

```
{
  "item": {
    "id": 1,
    "name": "Item 1"
  }
}
```

Provides specific id from Sample Data available i.e., URL: http://127.0.0.1:5000/items/1

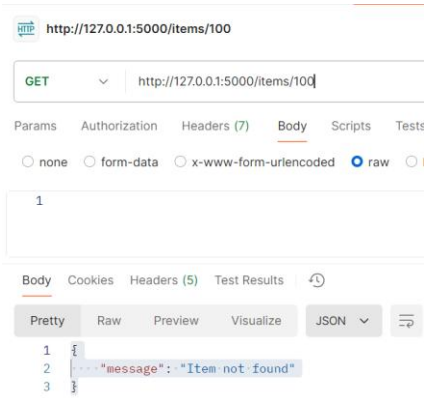
Demonstration:



Throws Error when not item not found from Sample Data. Example

URL: http://127.0.0.1:5000/items/100

Demonstration:



 # Endpoint to add a new item

Send a POST Request:

URL: http://127.0.0.1:5000/items

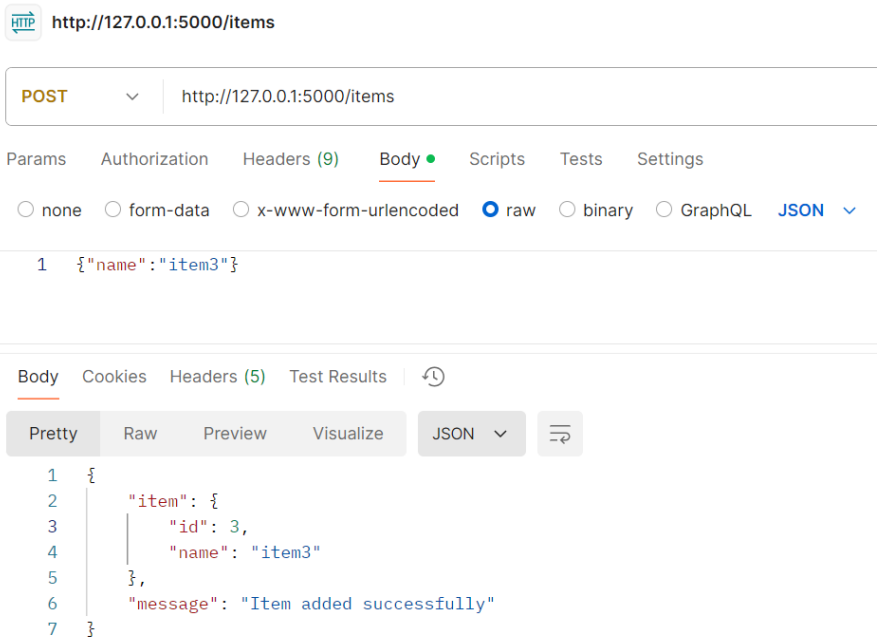
Body: Set the body to raw JSON and include:

{"name":"item3"}

Response:


```
{
   "item": {
      "id": 3,
      "name": "item3"
   },
   "message": "Item added successfully"
}
```


Demonstration:



Practical 4

Develop application to consume Google's search / Google's Map RESTful Web service.

To create a simple application that searches for a place using the Google Places API and retrieves the details from the Google Maps Geocoding API. Steps are as follows:

Software Tools Required

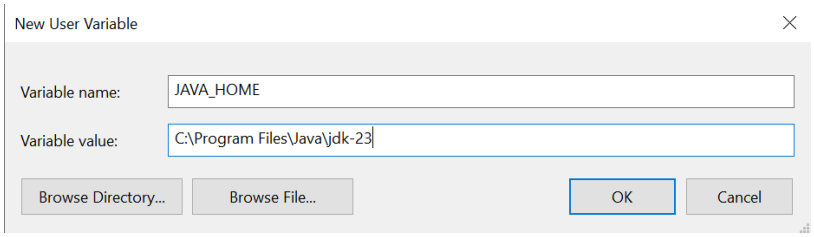
- **Code Editor:** VS Code
- **Gmail:** Before Starting this Practical, your personal Gmail account must be logged in the web browser.
- **Google Cloud SDK**: For managing Google APIs. Free Trial and Free Tier Services and Products | Google Cloud

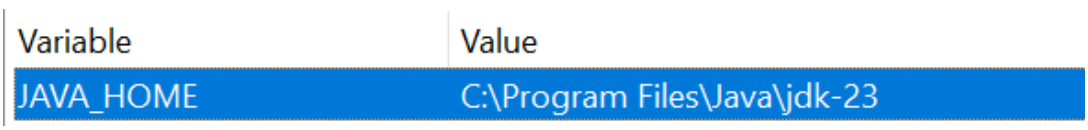
Downloads Required:

- JDK: Java Downloads | Oracle India (x64 MSI Installer)
- Apache Tomcat: Apache Tomcat Downloads. (x64 bit Windows zip)

After Downloading JDK, Set the Path in Environment Variables in User Variables > Path > New and Provide the Variable name and values to set the Path for JDK as shown below:

Demonstration:





Click OK to remaining opened Edit environment variable, Environment Variables and System Properties windows.

Now check the versions that shows path is set for Java JDK

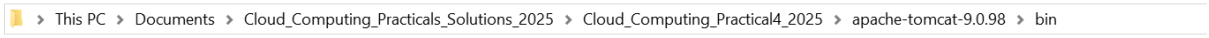
```
C:\Users\Kishore>java --version
java 23.0.1 2024-10-15
Java(TM) SE Runtime Environment (build 23.0.1+11-39)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.1+11-39, mixed mode, sharing)
```

Step 1: Download and Extract apache-tomcat-9.0.98 zip file into folder, and open apache-tomcat-9.0.98 folder till bin and select the directory path, cut (press backspace) and type **cmd.**

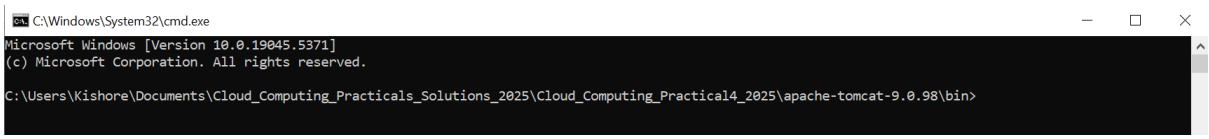
Demonstration:

Directory Path (Your directory path depends where you place your apache-tomcat folder)

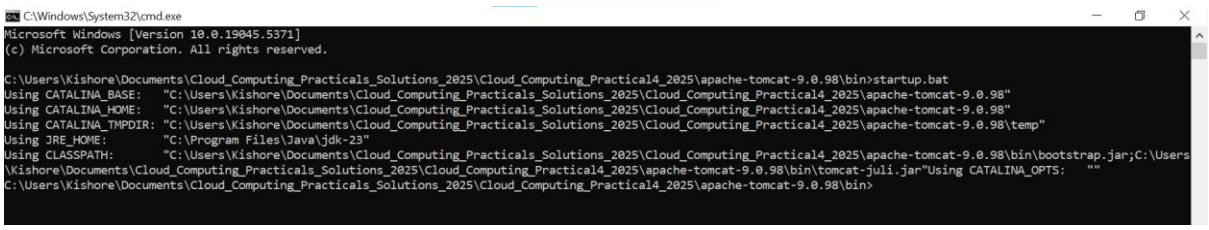
Select directory path



Type cmd and command prompt will open to the same directory path



Now type: **startup.bat**



We will get this tomcat loaded

Tomcat

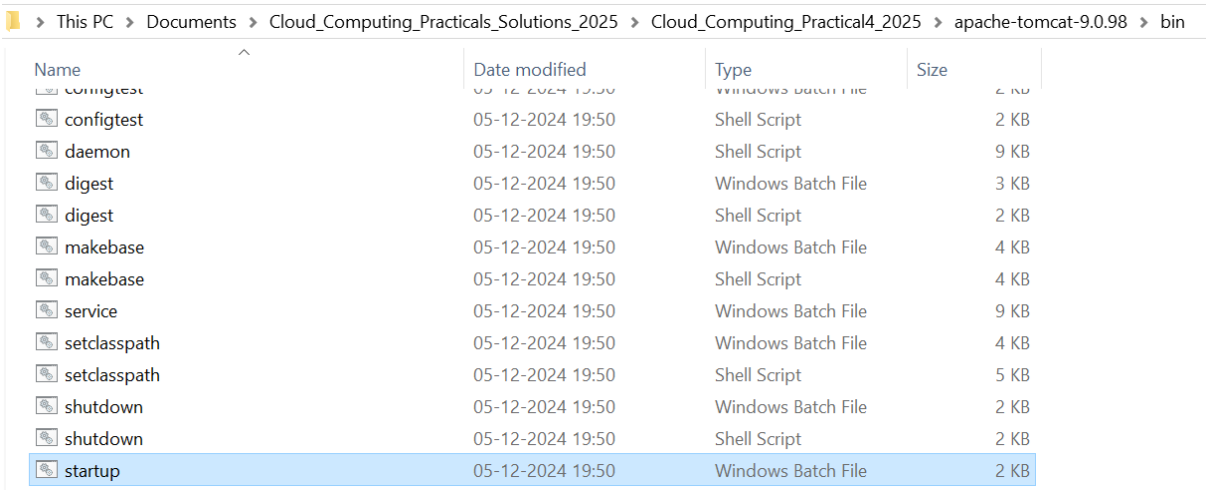
```
n directory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache
-tomcat-9.0.98\webapps\examples] has finished in [736] ms
18-Jan-2025 22:40:45.600 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application di
rectory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache-tom
cat-9.0.98\webapps\host-manager]
18-Jan-2025 22:40:45.676 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache
-tomcat-9.0.98\webapps\host-manager] has finished in [76] ms
18-Jan-2025 22:40:45.678 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application di
rectory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache-tom
cat-9.0.98\webapps\manager]
18-Jan-2025 22:40:45.768 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache
-tomcat-9.0.98\webapps\manager] has finished in [88] ms
18-Jan-2025 22:40:45.772 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application di
rectory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache-tom
cat-9.0.98\webapps\myjsp]
18-Jan-2025 22:40:45.835 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache
-tomcat-9.0.98\webapps\myjsp] has finished in [63] ms
18-Jan-2025 22:40:45.835 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application di
rectory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache-tom
cat-9.0.98\webapps\ROOT]
18-Jan-2025 22:40:45.903 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web applicatio
n directory [C:\Users\Kishore\Documents\Cloud_Computing_Practicals_Solutions_2025\Cloud_Computing_Practical4_2025\apache
-tomcat-9.0.98\webapps\ROOT] has finished in [68] ms
18-Jan-2025 22:40:45.919 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]

18-Jan-2025 22:40:45.966 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in [1694] milliseconds
```

Note:

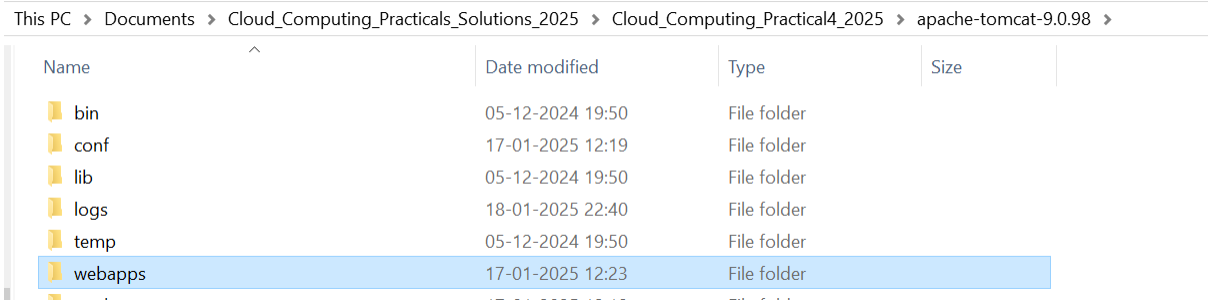
This Tomcat Server must be kept running in the background by minimizing this window. Otherwise, the we won't get the output.

Another way to load Tomcat

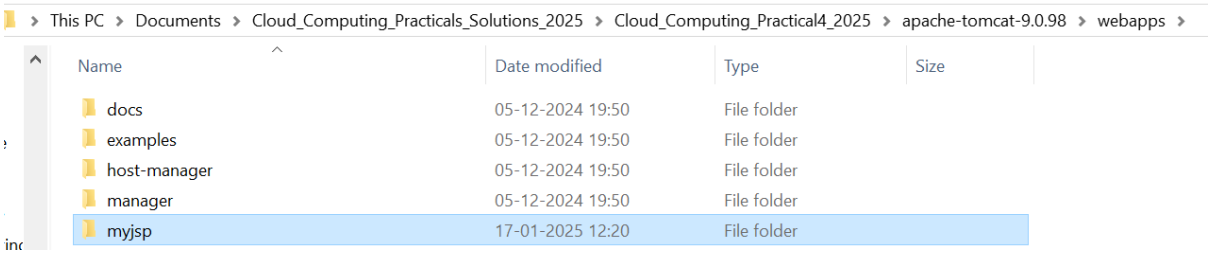
Also, in bin folder we will have startup as Windows Batch File, double click Run and it will load Tomcat.



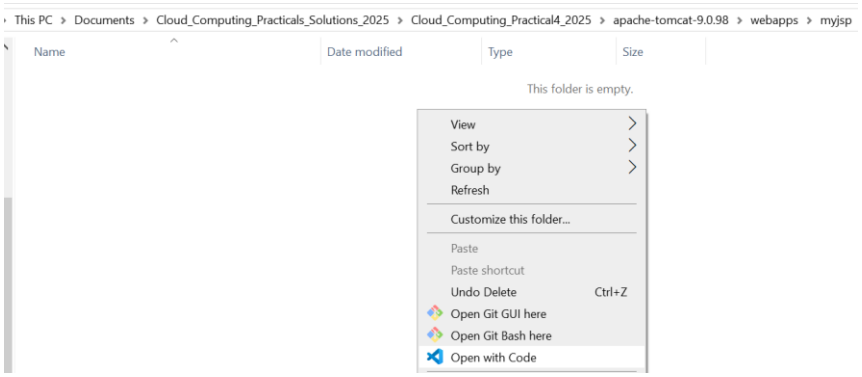
Step 2: After completing Step 1, Now open apache-tomcat-9.0.98 folder, open a folder name **webapps** which is already created in apache-tomcat-9.0.98 folder



Open **webapps** folder, create a new folder and name it as: **myjsp**



Step 3: Open **myjsp** folder and open it with VS Code



Create 2 jsp files with name **myindex.jsp** and **myinput.jsp**

1. **Filename:** myindex.jsp : **Display Map with Google Maps API**

Code:

```
<%@ page contentType="text/html" pageEncoding="UTF-8" %>

<!DOCTYPE html>

<html>

<head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Google Maps Location</title>
    <!-- Google Maps JavaScript API -->
    <script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY
&callback=initMap" async defer></script>
    <style>
        /* Set the size of the map container */
        #map {
            height: 400px;
            width: 100%;
        }
    </style>
</head>




<body>
<%
    // Fetch latitude and longitude from form inputs
    String latParam = request.getParameter("t1");
    String longParam = request.getParameter("t2");
    // Default values in case parameters are not provided
    double lati = 40.7128;  // Default latitude (New York)
    double longi = -74.0060; // Default longitude (New York)
    // If parameters exist, parse them to doubles
    if (latParam != null && !latParam.trim().isEmpty()) {
        lati = Double.parseDouble(latParam);
    }
    if (longParam != null && !longParam.trim().isEmpty()) {
        longi = Double.parseDouble(longParam);
```

```
    }
%>
<h3>Google Maps Location</h3>
<div id="map"></div>
<script>
    // Initialize the map
    function initMap() {
        var location = { lat: <%= lati %>, lng: <%= longi %> };  // Use JSP values for lat and lng
        // Create a new map centered on the location
        var map = new google.maps.Map(document.getElementById('map'), {
            zoom: 10,  // Zoom level
            center: location  // Center the map on the provided coordinates
        });



        // Add a marker at the location
        var marker = new google.maps.Marker({
            position: location,
            map: map
        });
    }
</script>
</body>
</html>
```

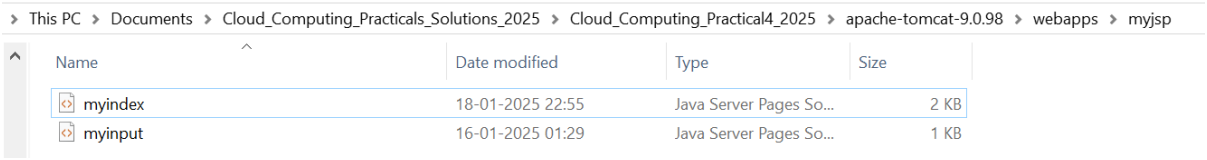
2. **Filename**: myinput.jsp : **Form to Get Latitude and Longitude**

Code:

```
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
```

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Enter Latitude and Longitude</title>
</head>
<body>
    <!-- Form to accept input values for latitude and longitude -->
    <form action="myindex.jsp" method="get">
        <pre>
            Enter latitude: <input type="text" name="t1" />
            Enter longitude: <input type="text" name="t2" />
            <input type="submit" value="Show" />
        </pre>
    </form>
</body>
</html>
```

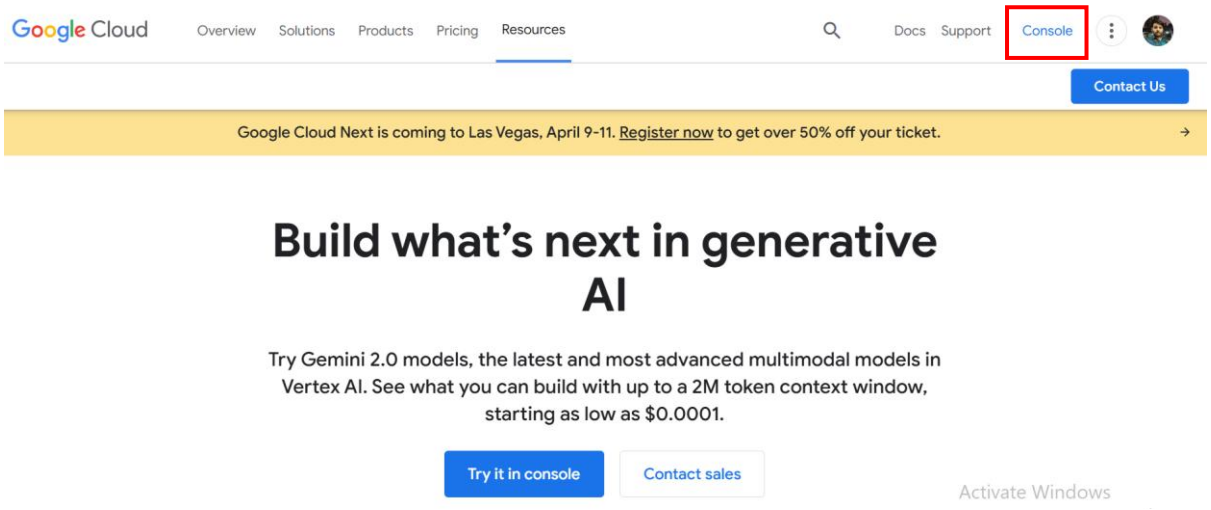
Your **myjsp** folder will look like this



Step 4: Open Google Cloud Platform Console,

Free Trial and Free Tier Services and Products | Google Cloud

Click on Console

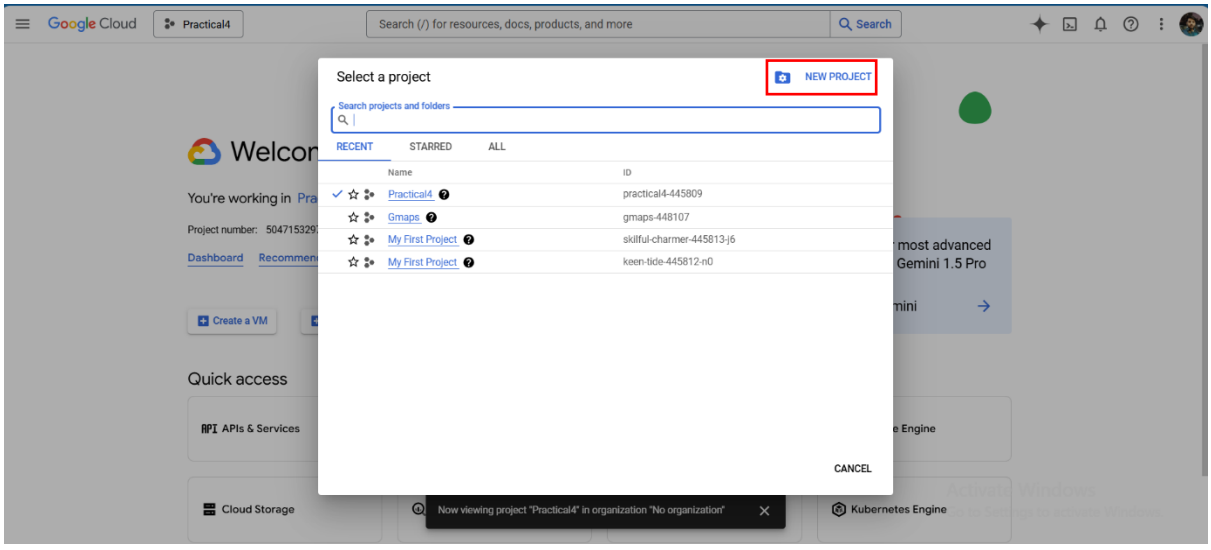


Step-by-Step Guide to Implement Google Maps API with JSP

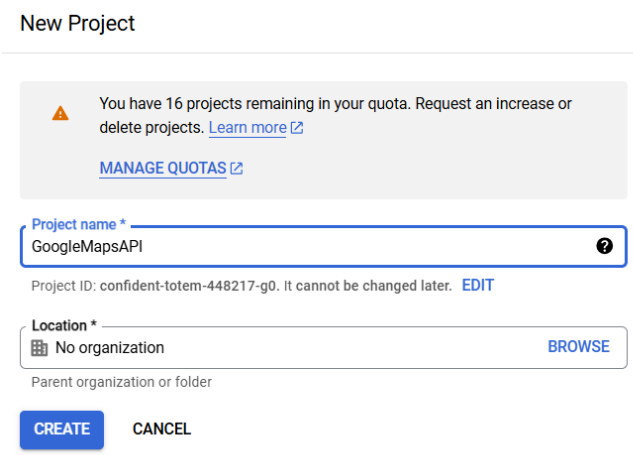
1. Get Your Google Maps API Key

- First, you need to obtain an API key from **Google Cloud Console** to use the **Google Maps JavaScript API**. Follow these steps:
    - Go to Google Cloud Console.
    - Create a new project (or select an existing one).
    - Enable the **Maps JavaScript API**. (Instead of Places API and Geocoding API)
    - Under **Credentials**, create an **API Key** and copy it.

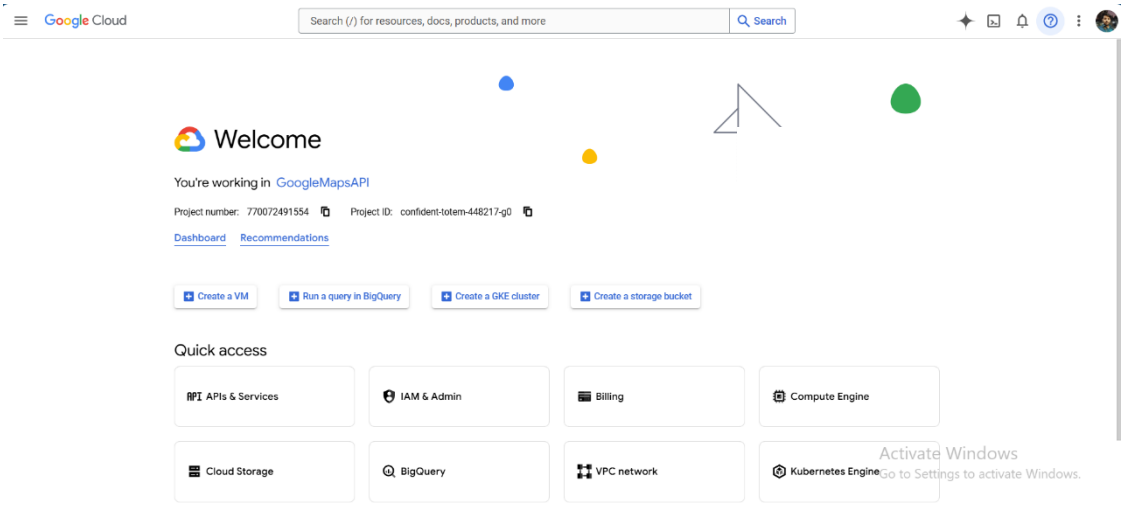
Demonstration: Here we have existing projects, now click on **NEW PROJECT**



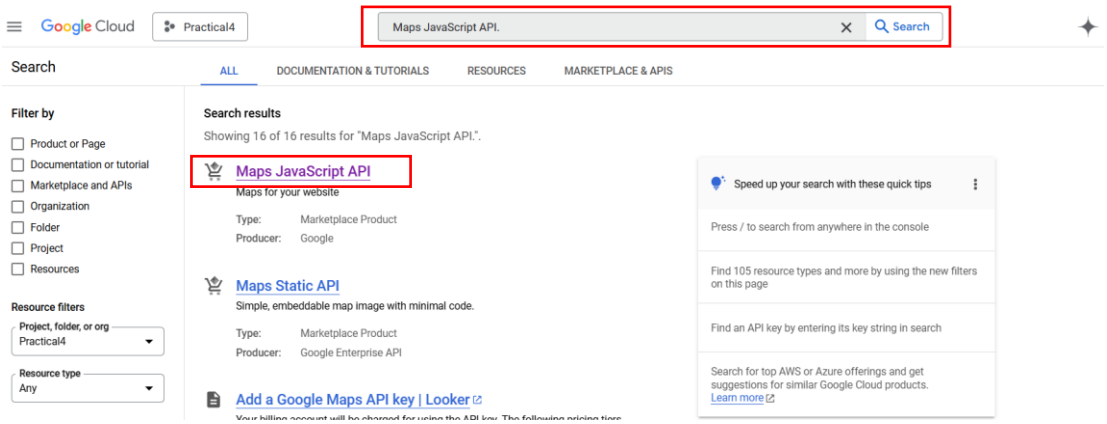
Project Name: GoogleMapsAPI and click CREATE

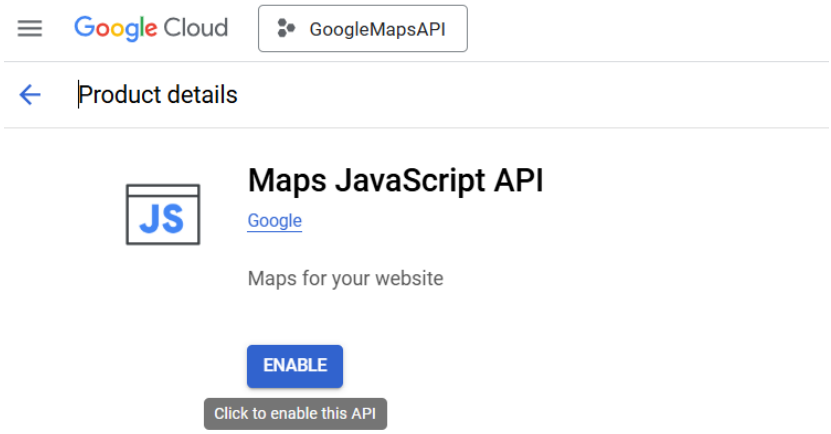


Now we can see our new project GoogleMapsAPI and Notifications of new project

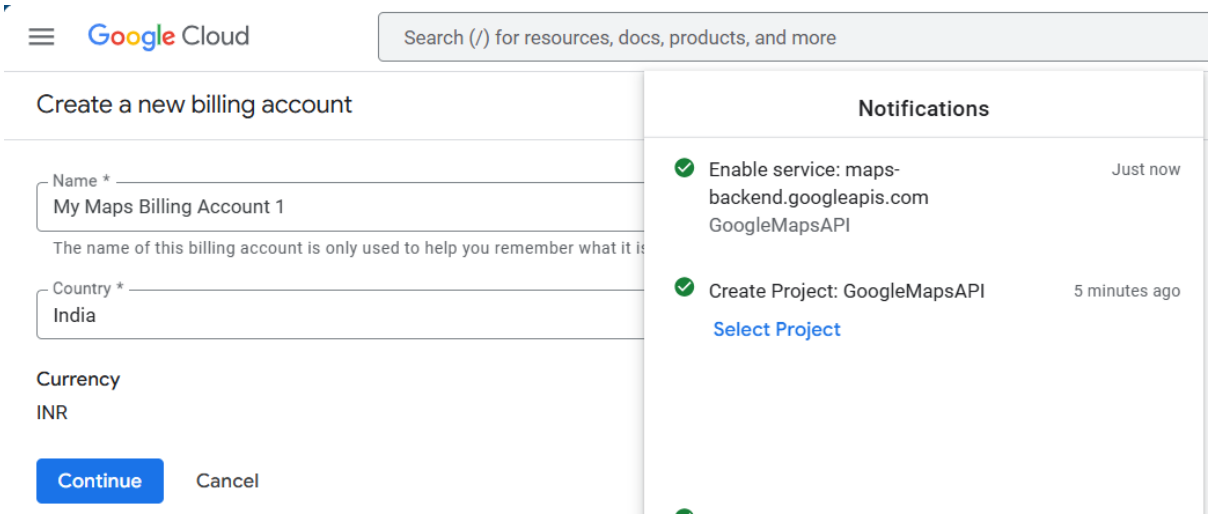


Search, Click and Enable the **Maps JavaScript API**.

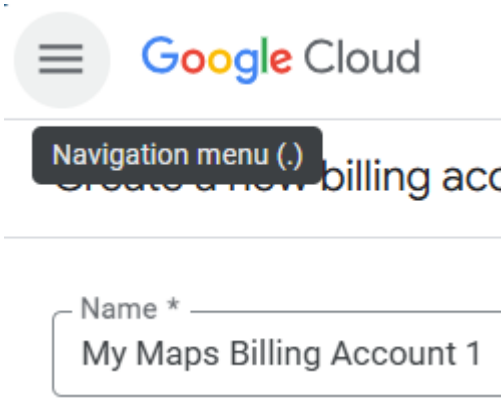




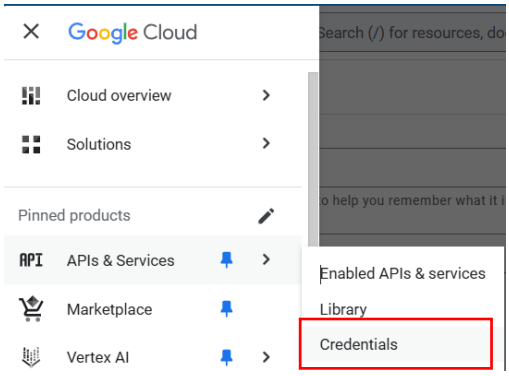
After Enabling Maps JavaScript API, it will show notifications and also redirect to billing method (Ignore billing method)



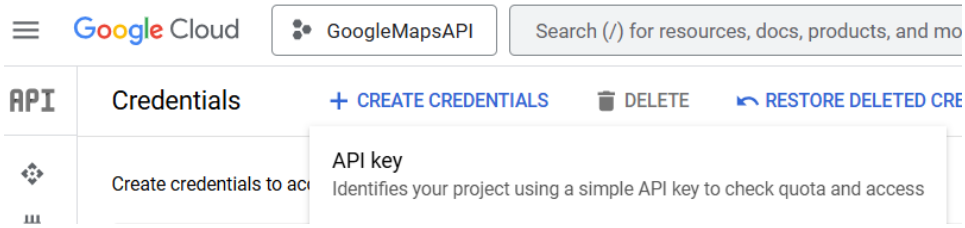
Now click on Navigation Menu

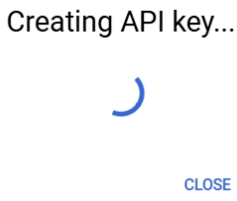


APIs & Services > Credentials

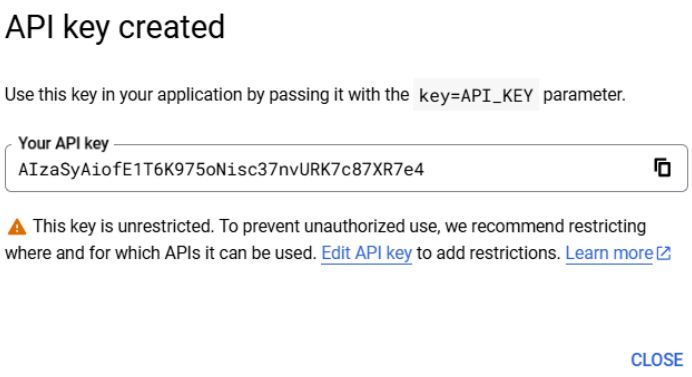


Click + CREATE CREDENTIALS > API Key





We got the API Key Created, Copy the API Key and paste it in **myindex.jsp** file.



Now, open **myjsp** folder and open with VS Code and paste the API_KEY in **myindex.jsp** script tag:

Shown below:

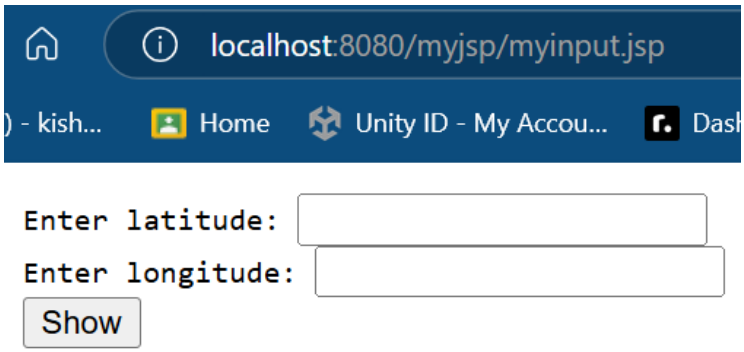
Before:<!-- Google Maps JavaScript API -->

<script src="https://maps.googleapis.com/maps/api/js?key=**YOUR_API_KEY**&callback=initMap" async defer></script>

After: :<!-- Google Maps JavaScript API --><script src="https://maps.googleapis.com/maps/api/js?key=**AIzaSyAiofE1T6K975oNisc37nvURK7c87XR7e4** &callback=initMap" async defer></script>

Step 5: Now to check the output, open a web browser Google Chrome or Microsoft Edge Ctrl + Click to follow the Link, URL: http://localhost:8080/myjsp/myinput.jsp

Demonstration:



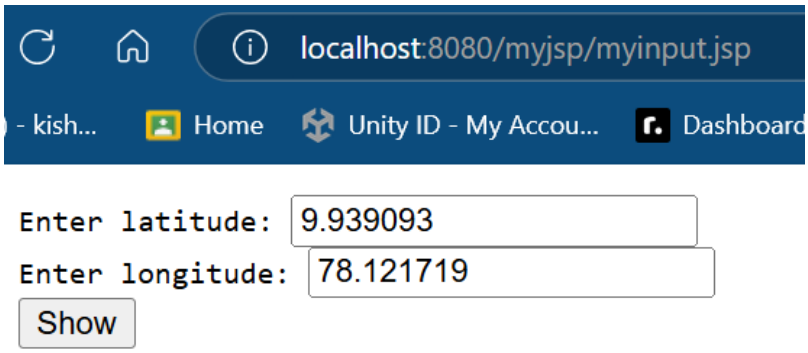
Enter Example latitude and longitude of any place

Here, latitude and longitude coordinates of city: Madurai

Latitude: 9.939093

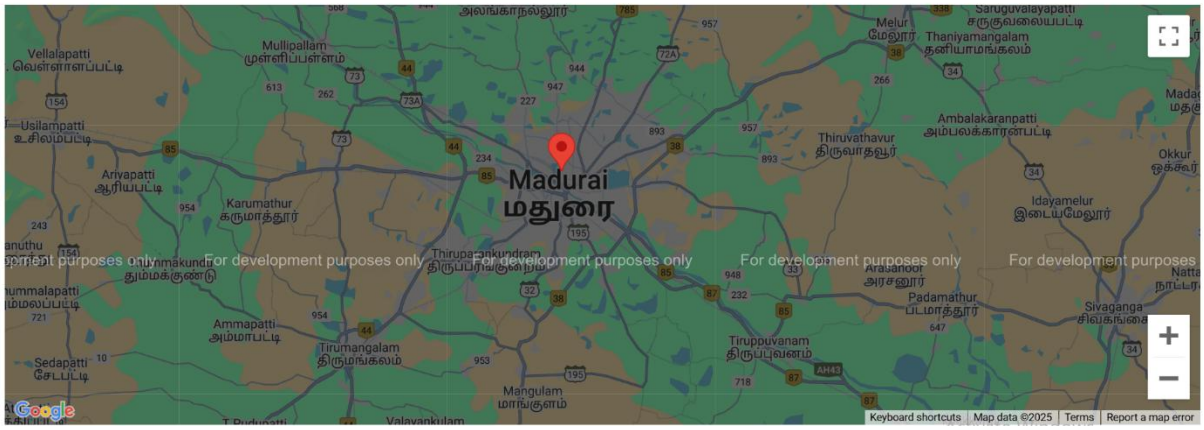
Longitude: 78.121719

Output: **Enter co-ordinates and click on Show**



It will redirect to the Google Maps: Exact Coordinates Location

Google Maps Location



Practical 5

Installation and Configuration of virtualization using KVM.

Software Tools Required

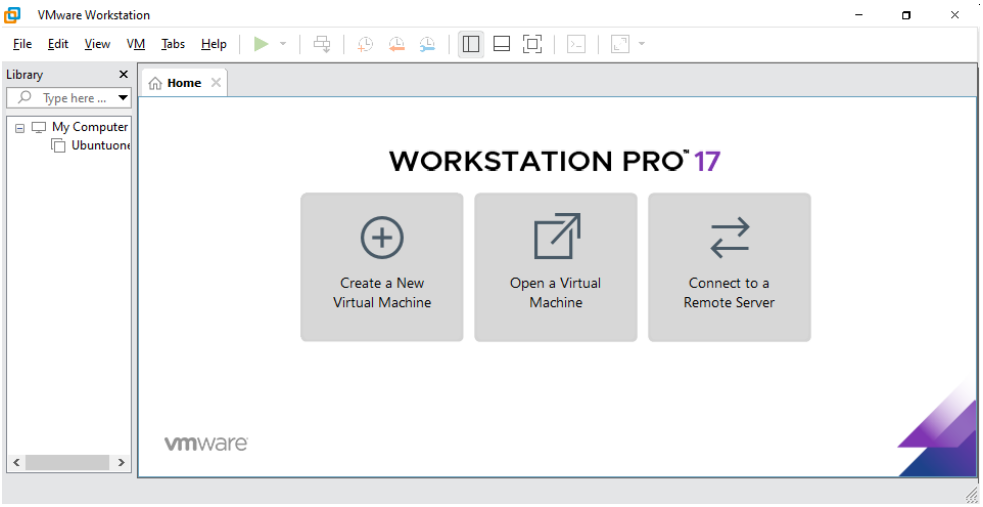
- **Virtual Machine Monitor:** VMware Workstation
- **Operating System:** Linux

Downloads Required:

- Virtual Machine Monitor**:** Download VMware Workstation Pro 17.6.0 Build 24238078 for Windows | Uptodown.com
- Operating System: Download Ubuntu Desktop | Ubuntu (for Linux distribution of your choice)

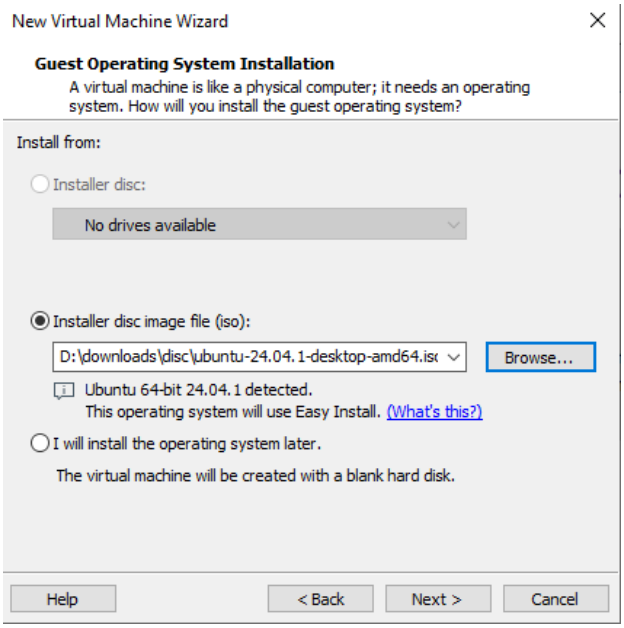
Open Vmware Workstation and start the steps

Demonstration:

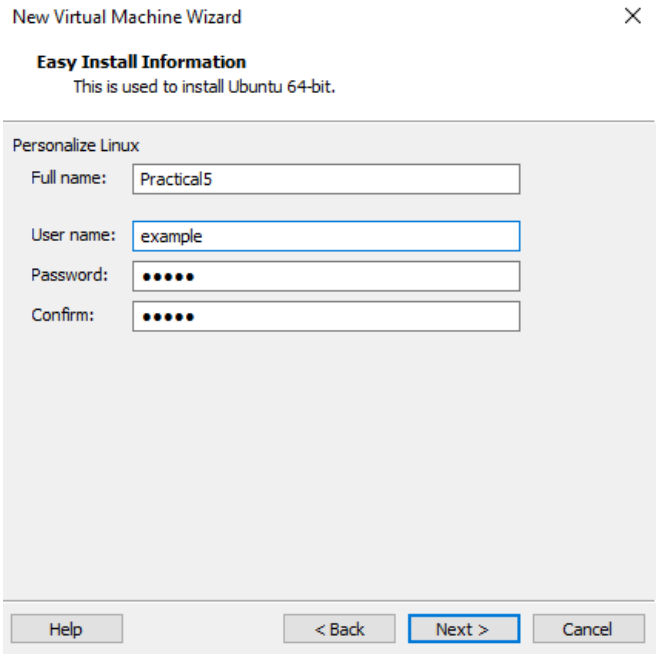


Create a New Virtual Machine >

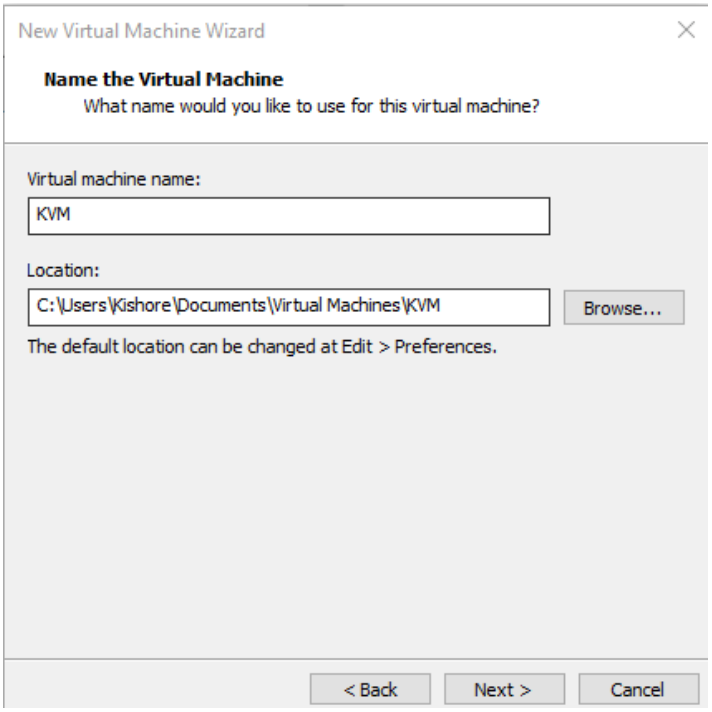
Browse and place the ISO File



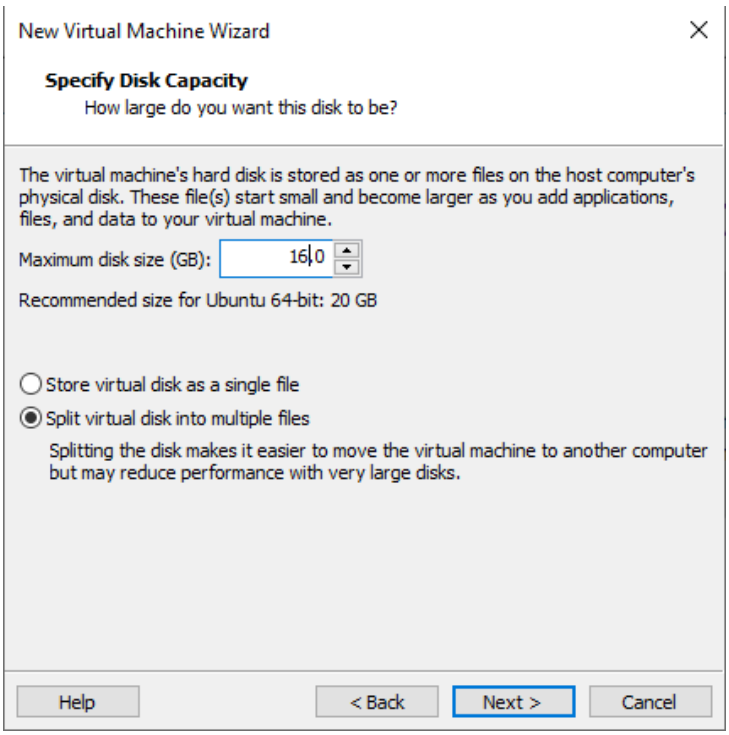
Provide full name as **Practical5** and user name as **example** and password and confirm password as **linux**



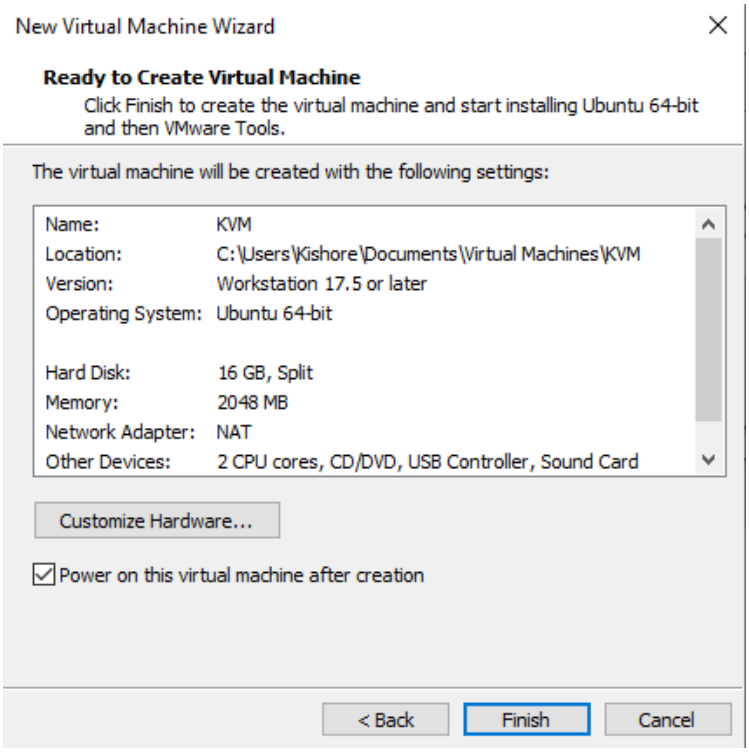
Give virtual machine name as **KVM**



Provide disk size: 16.0GB then click Next



Click on Customize Hardware



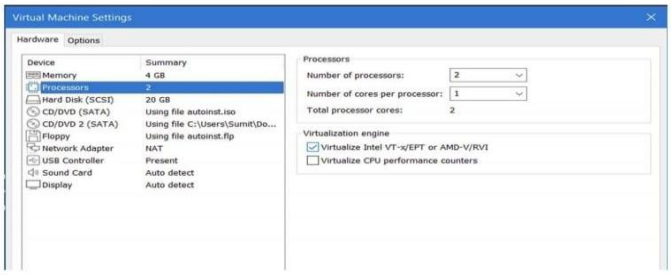
To enable KVM

Check the Virtualize Intel VT

For enabling KVM

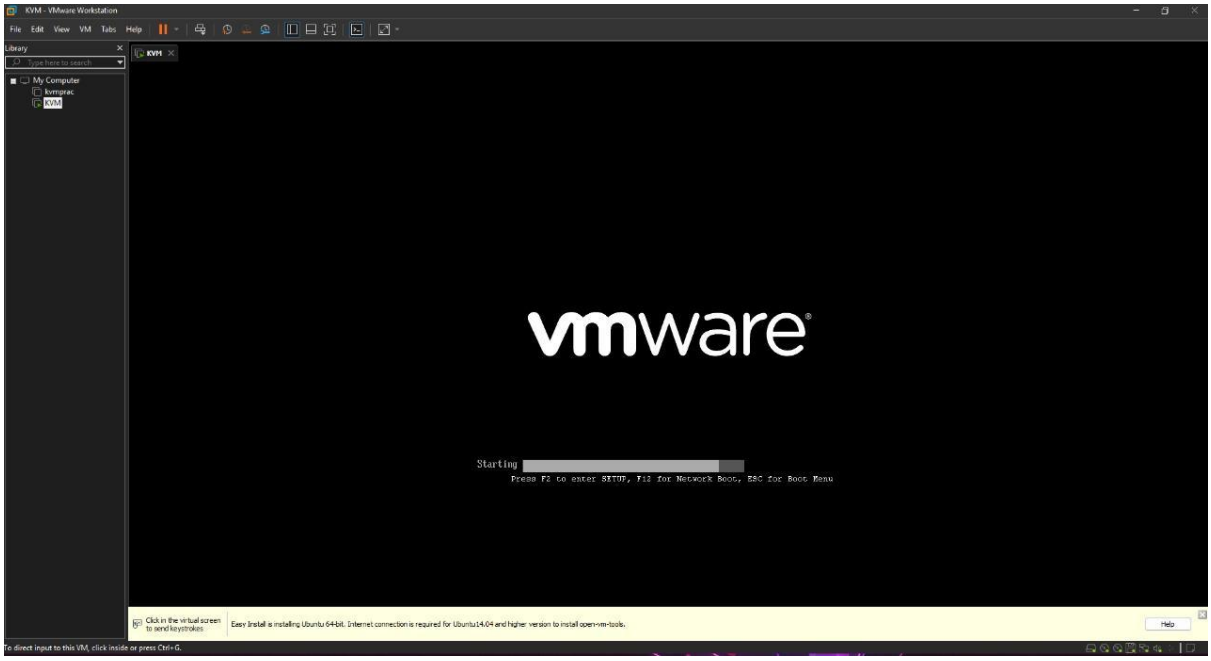
1.Edit virtual machine settings

2.Select processor then tick the Virtualize Intel VT

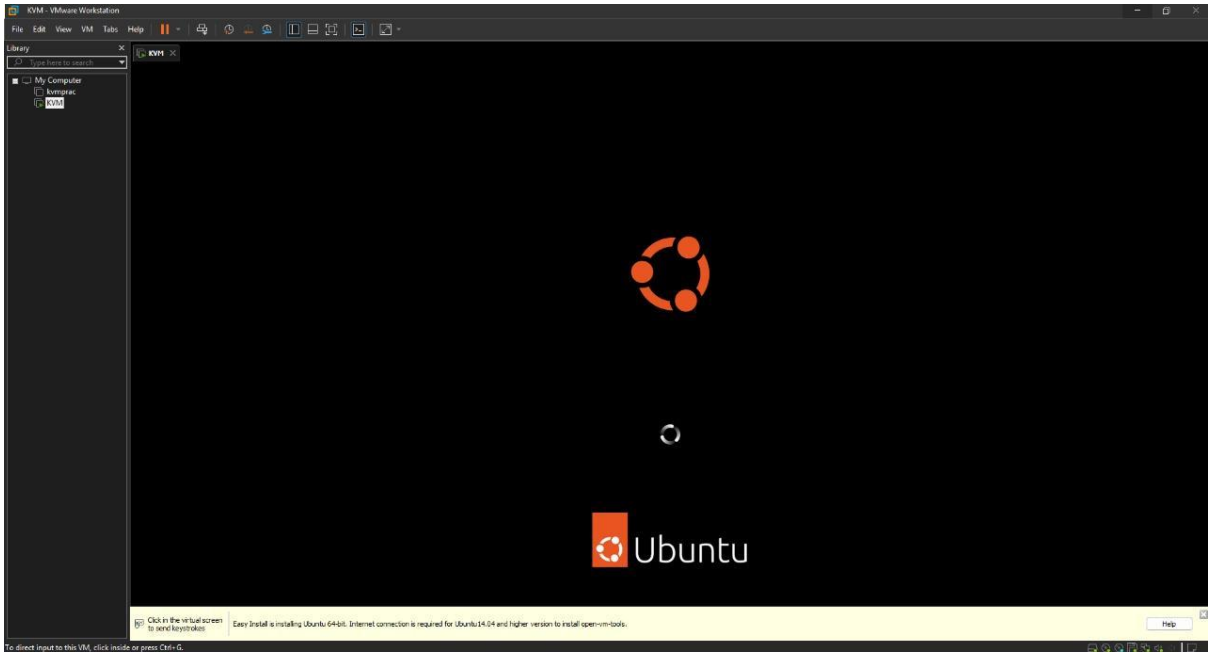


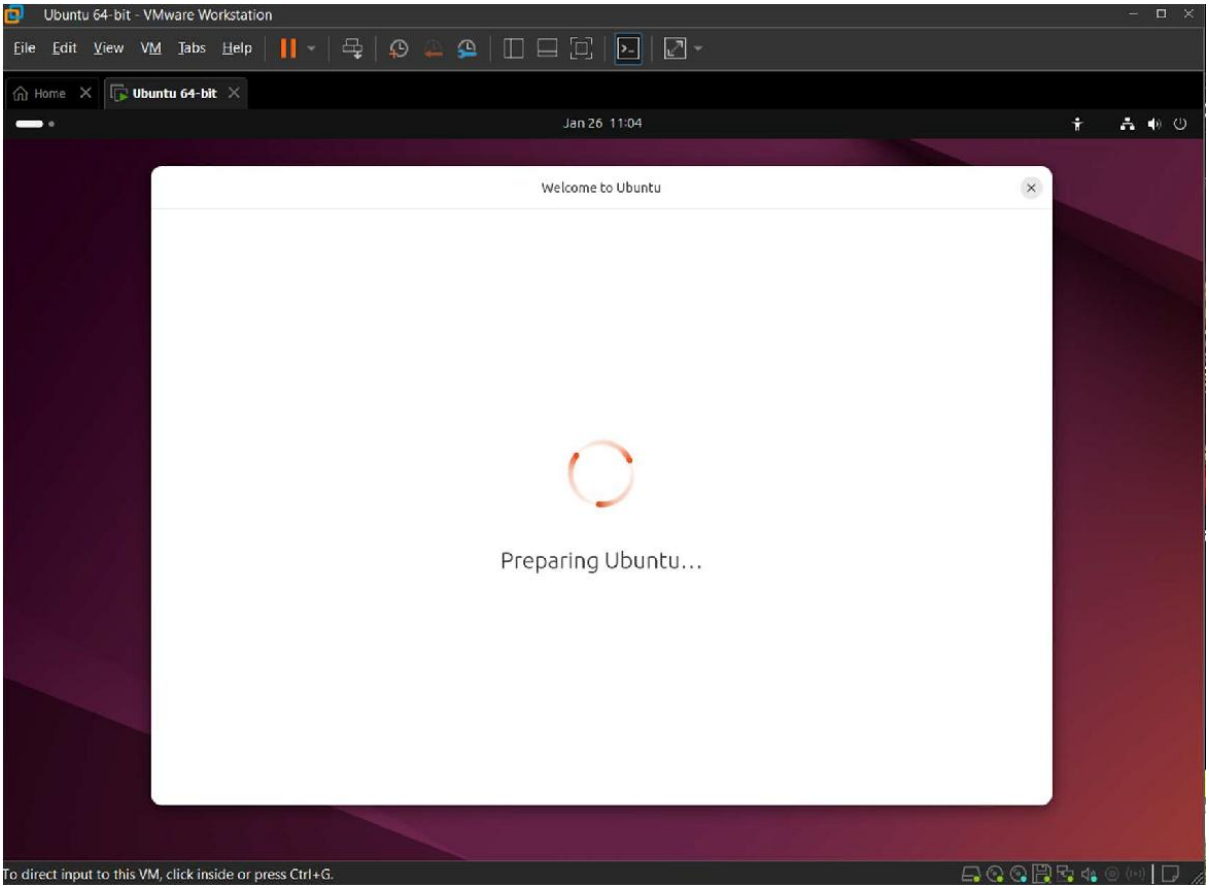
Select OK

(In case You get an error like Intel VT not supported, Turn off the Memory Integrity from core isolation)



Ubuntu, a linux Distribution system will load.





Ubuntu Setup:

1.Choose Language then click next

2.Then in Accessibility Tab Click next

3.Keyboard Layout As English US then next

4.Connect to Internet then next

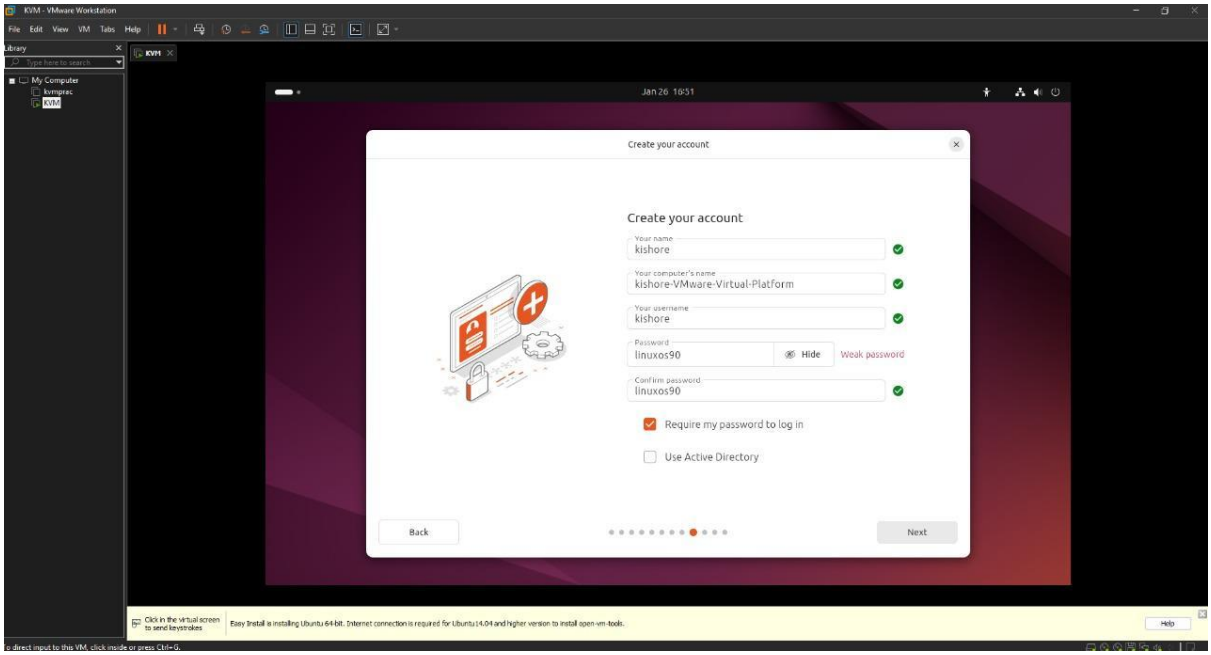
5.Tick Install ubuntu then next

6.Interactive then Default Installation

7.Install Proprietary Software (tick both options)

8.Then Tick Erase disk and install ubuntu

9.Then Fill up the following details:

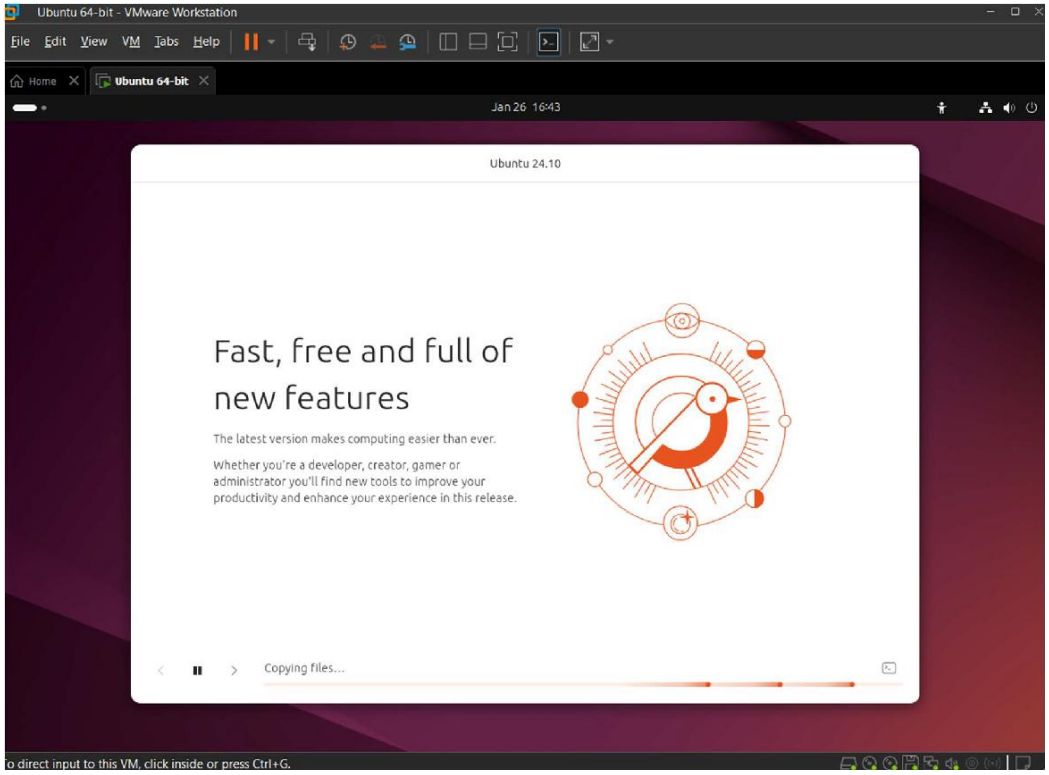


10.Then set the time zone and click next

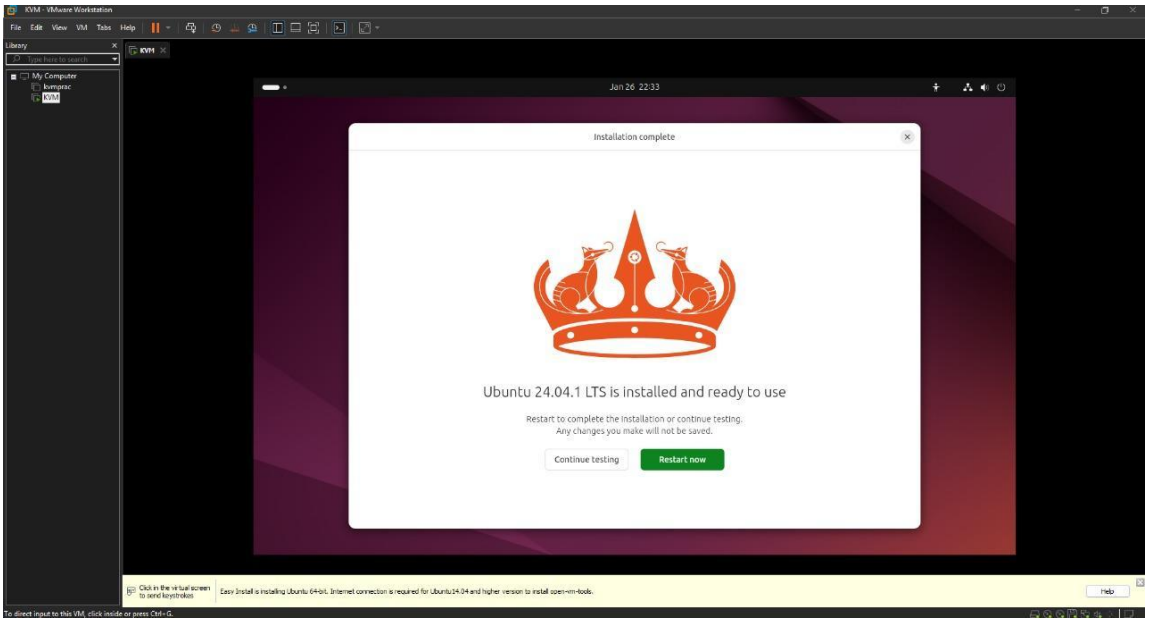
Location: Mumbai

Timezone:Asia/Kolkata

11.Then Following window appears, Click Install



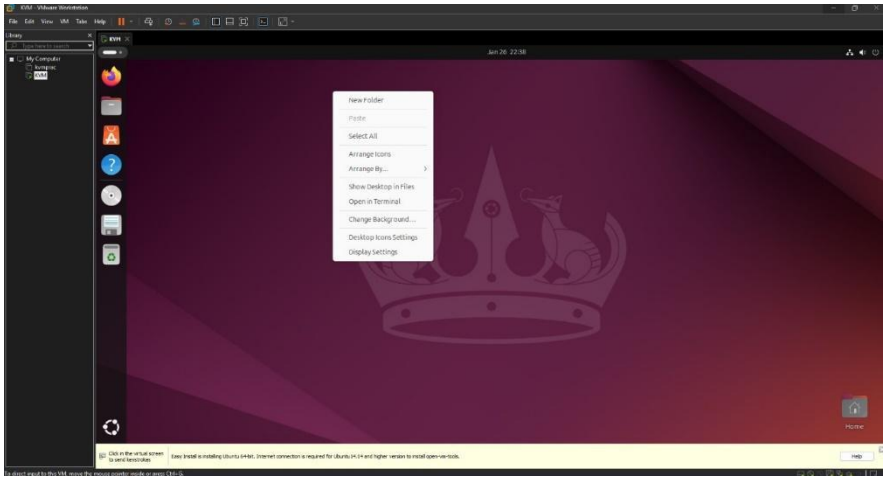
We get the following screen,



Click Restart.

Now login using your password.

II)KVM Setup Steps: Open Terminal



1. Update the System

sudo apt update && sudo apt upgrade -y

2. Check if Virtualization is Enabled:
sudo grep -c "svm\|vmx" /proc/cpuinfo

3. Verify KVM Virtualization

Check if KVM virtualization is enabled by running command:
kvm-ok

If the kvm-ok command is not found, install the CPU checker tool:

sudo apt install cpu-checker

then type kvm-ok

Output should include:

INFO: /dev/kvm exists

KVM acceleration can be used.

4. Install KVM and Required Packages

sudo apt install qemu-kvm virt-manager libvirt-daemon-system libvirt-clients bridge-utils -y

5. Enable the Virtualization Daemon

Start and enable the libvirt daemon:

sudo systemctl enable libvirtd

sudo systemctl start libvirtd

6. Check the Status of the Libvirt Daemon

Verify that the daemon is running:

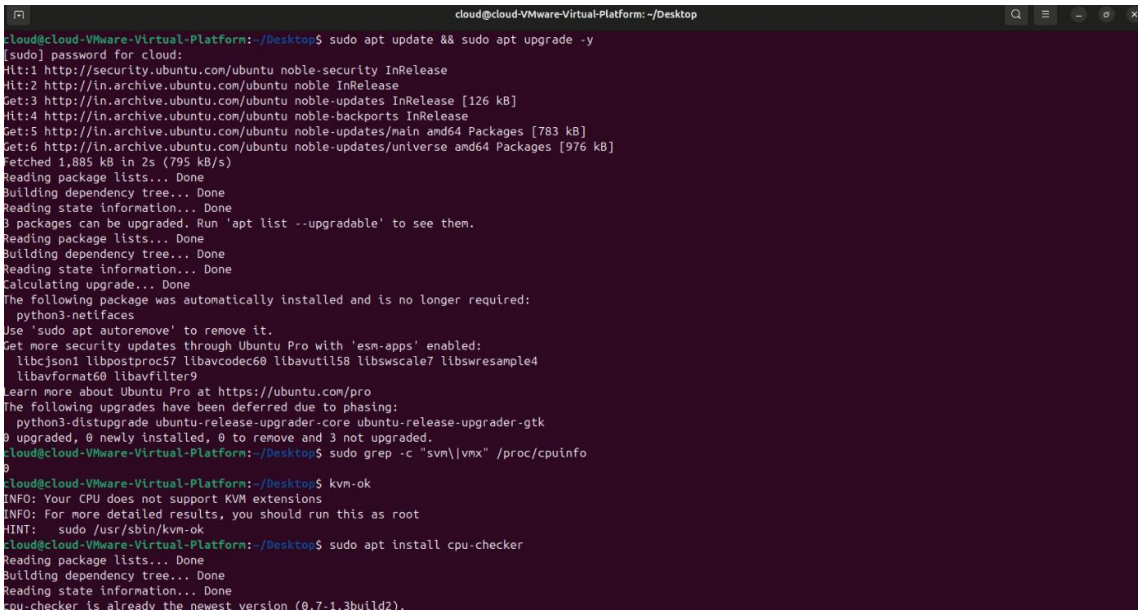
sudo systemctl status libvirtd

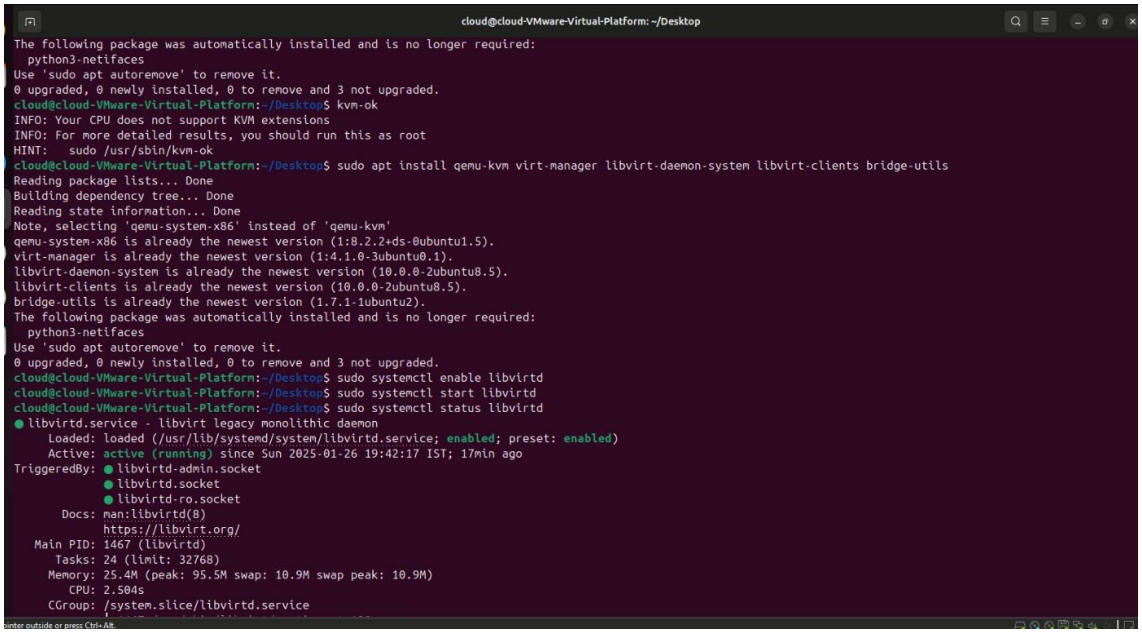
7. Add Your User to KVM and Libvirt Groups

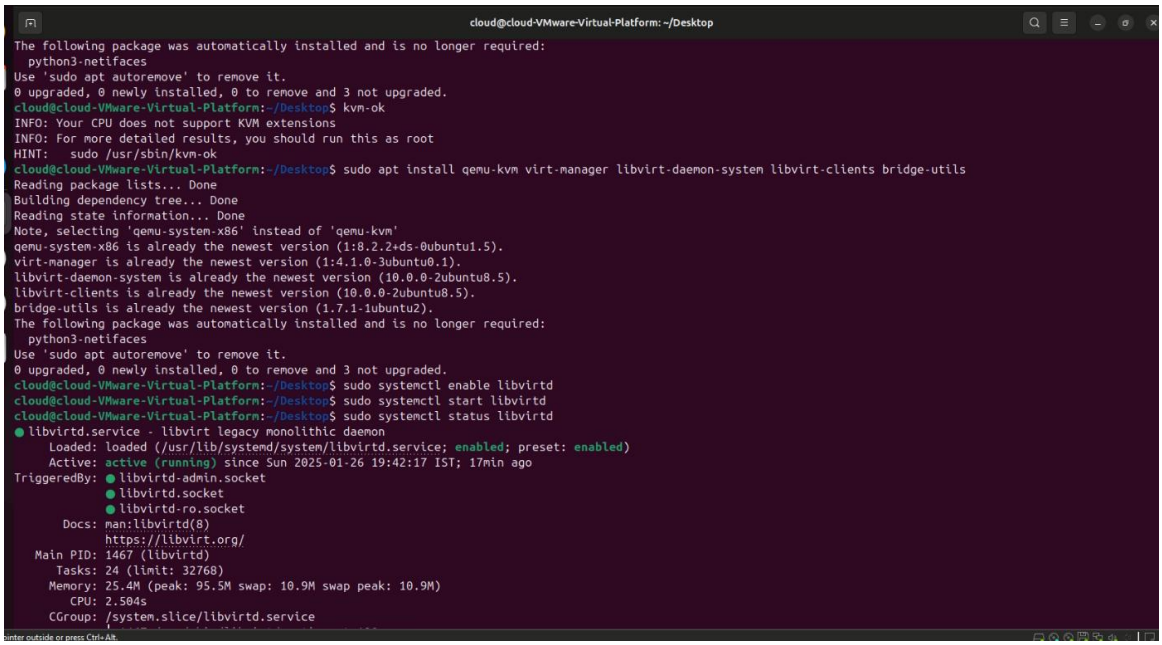
Replace your-username with your actual username and run the following commands:

sudo usermod -aG kvm your-username

sudo usermod -aG libvirt your-username







8. Log Out and Re-login

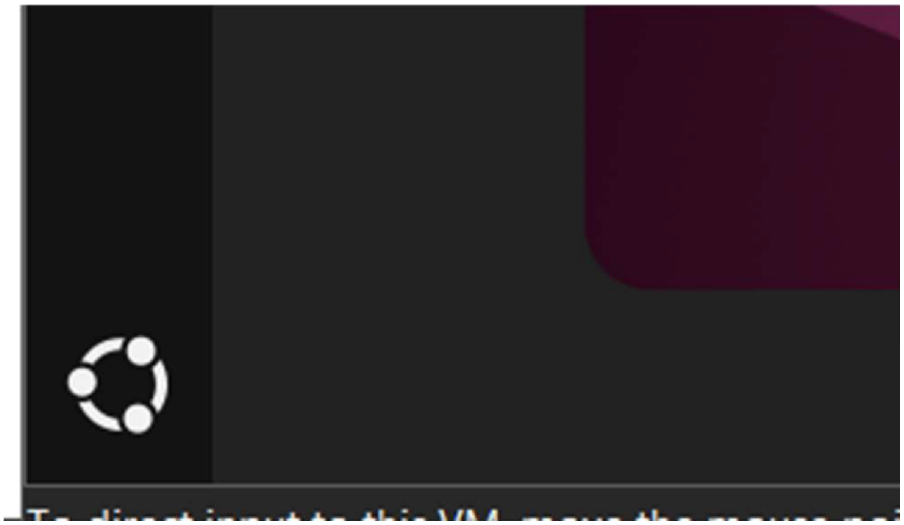
Log out from your system and log back in to apply group changes.

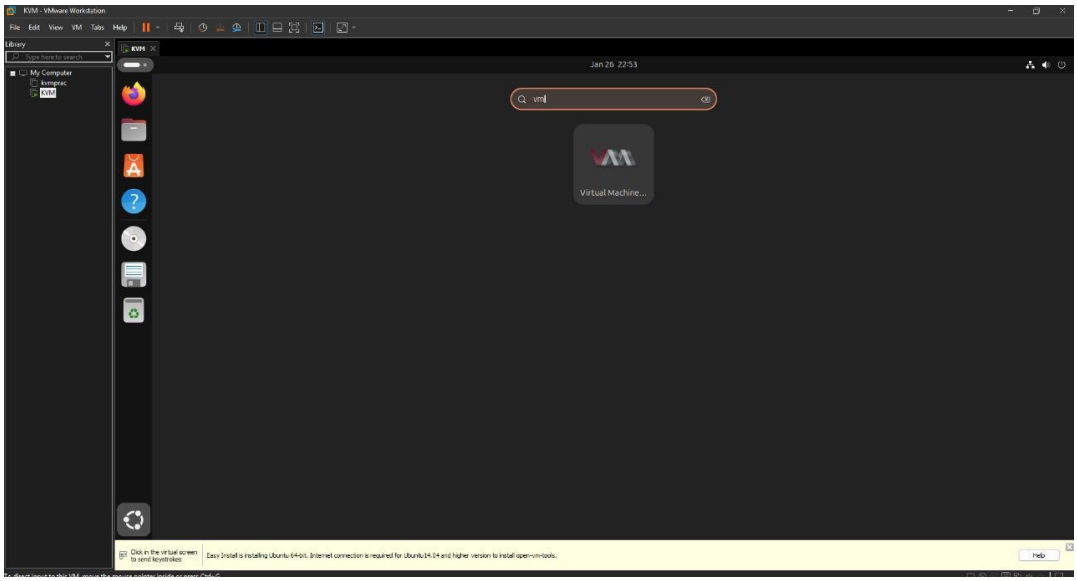
9. Run KVM Virtual Machine Manager

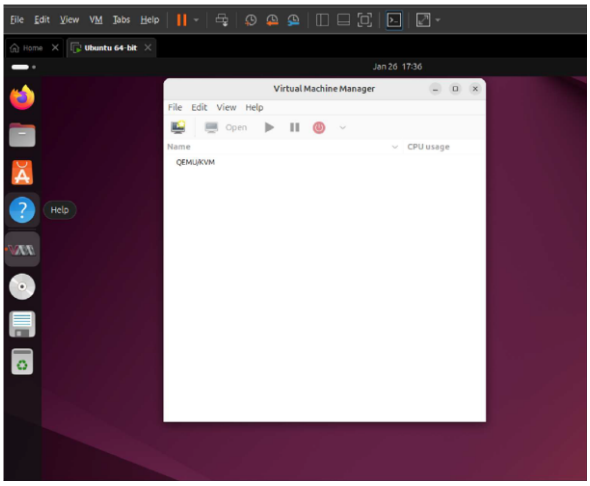
Search for "Virtual Machine Manager" in your system applications and launch it.

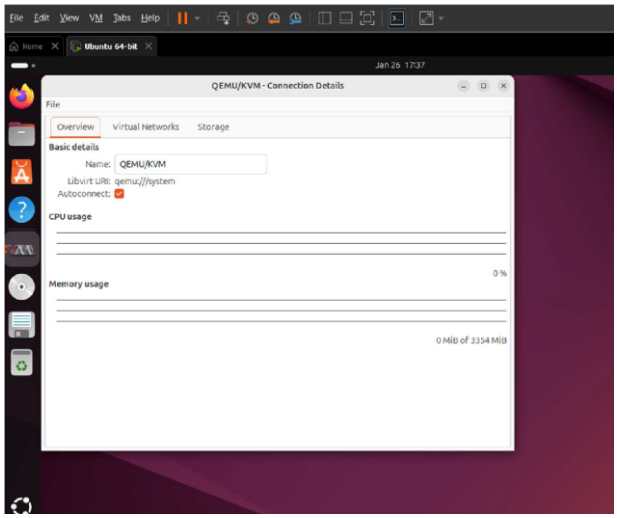
10. Prepare to Create Virtual Machines

In Virtual Machine Manager, click Create a New Virtual Machine to begin setting up virtual environments.

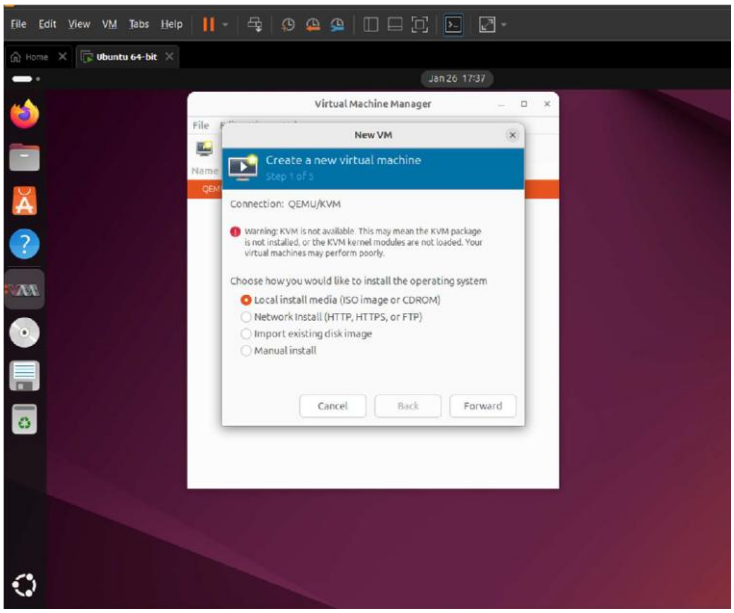



KVM - VMware Workstation
Jan 26 22:53
vmi
Virtual Machine...

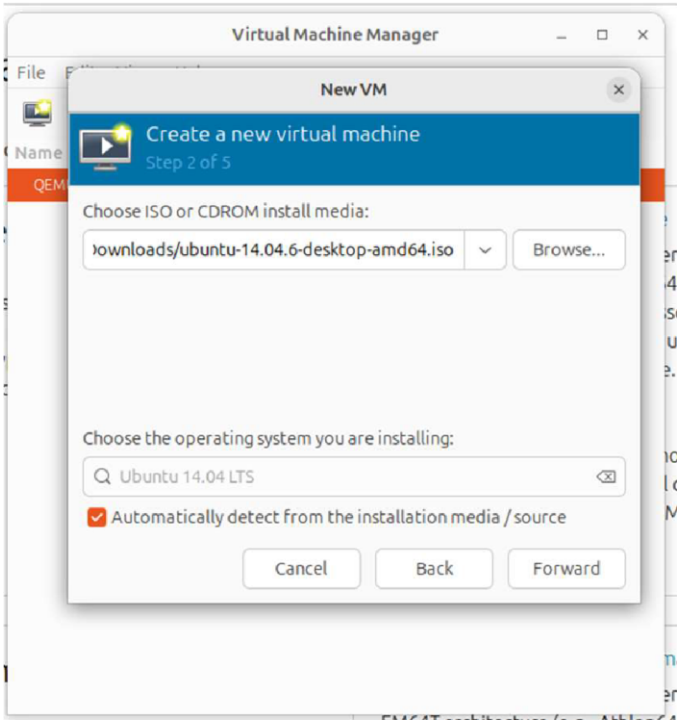

Ubuntu 64-bit
Jan 26 17:36
Virtual Machine Manager
File Edit View Help
Open
Name
CPU usage
QEMU/KVM
Help


Ubuntu 64-bit
Jan 26 17:37
QEMU/KVM - Connection Details
File
Overview
Virtual Networks
Storage
Basic details
Name: QEMU/KVM
Libvirt URI: qemu:///system
Autoconnect:
CPU usage
0 %
Memory usage
0 MiB of 3354 MiB

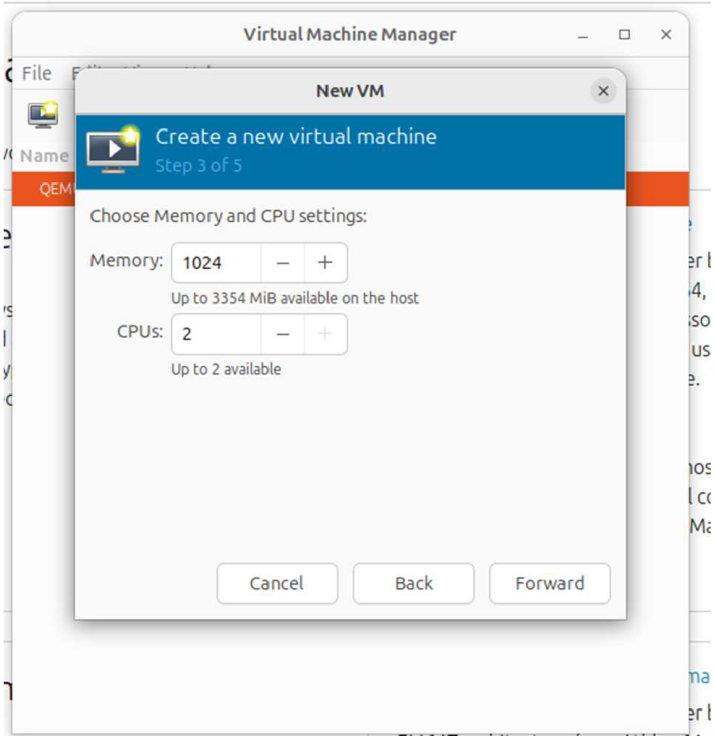
Now we can create new virtual machine from File>Create New Virtual Machine



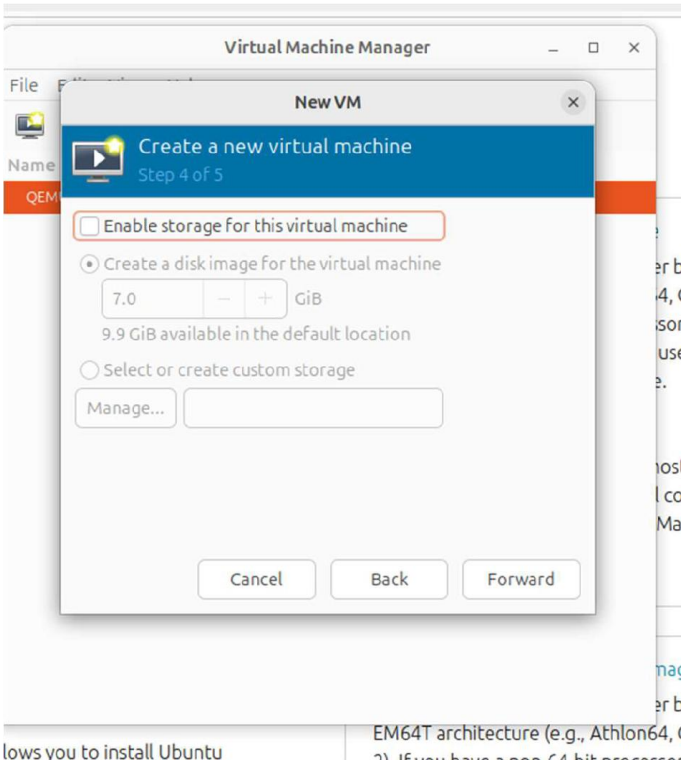
Then ,Select the os to install (eg/:windows, ubuntu etc)

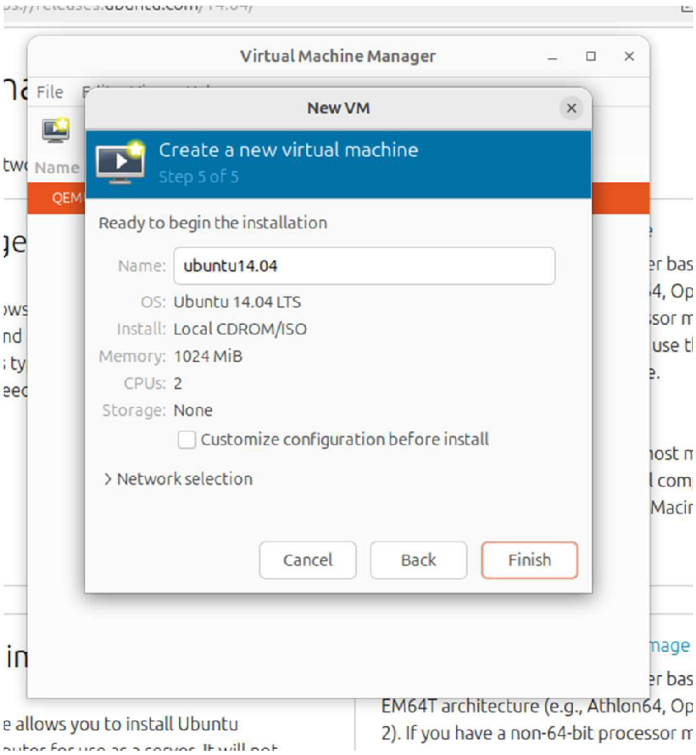


Then

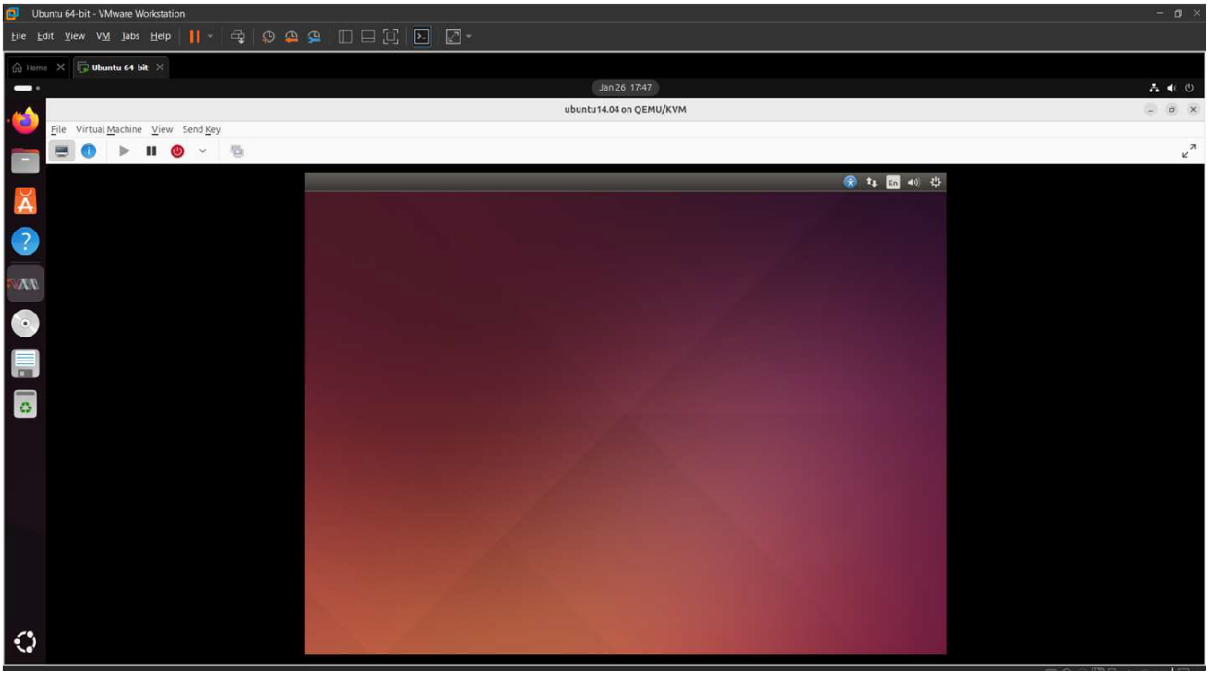

Virtual Machine Manager
File
New VM
Create a new virtual machine
Step 3 of 5
Choose Memory and CPU settings:
Memory: 1024
Up to 3354 MiB available on the host
CPUs: 2
Up to 2 available
Cancel
Back
Forward

Then


Virtual Machine Manager
File
New VM
Create a new virtual machine
Step 4 of 5
Enable storage for this virtual machine
Create a disk image for the virtual machine
7.0
GiB
9.9 GiB available in the default location
Select or create custom storage
Manage...
Cancel
Back
Forward
EM64T architecture (e.g., Athlon64,


Virtual Machine Manager
File
New VM
Create a new virtual machine
Step 5 of 5
Ready to begin the installation
Name: ubuntu14.04
OS: Ubuntu 14.04 LTS
Install: Local CDROM/ISO
Memory: 1024 MiB
CPUs: 2
Storage: None
Customize configuration before install
Network selection
Cancel
Back
Finish
e allows you to install Ubuntu
EM64T architecture (e.g., Athlon64, Op
2). If you have a non-64-bit processor m


QEMU/KVM
ubuntu14.04
Running


Ubuntu 64-bit - VMware Workstation
File Edit View VM Tabs Help
Ubuntu 64-bit
Jan 26 17:47
ubuntu14.04 on QEMU/KVM
File Virtual Machine View Send Key

Practical 6

Develop application to download image/video from server or upload image/video to server using MTOM techniques

Software Tools Required

- **IDE or Code Editor:** Netbeans version 8.0.2
- **JDK:** java jdk version 8

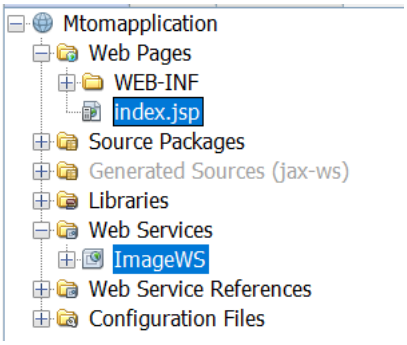
Downloads Required:

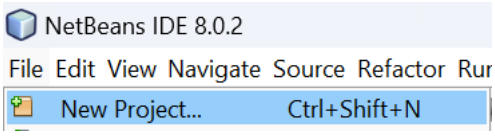
- Netbeans 8.0.2: https://dlc-cdn.sun.com/netbeans/8.0.2/final/bundles/netbeans-8.0.2-windows.exe
- Java JDK 8: https://www.oracle.com/in/java/technologies/javase/javase8u211-later-archive-downloads.html#license-lightbox

jdk1.8.0_431-20250305T110059Z-001.zip

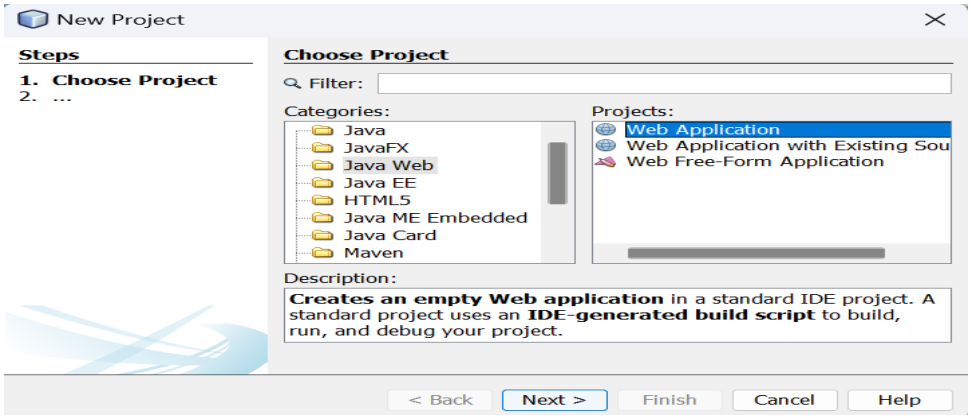
Project Name: Mtomapplication

Project Architecture:

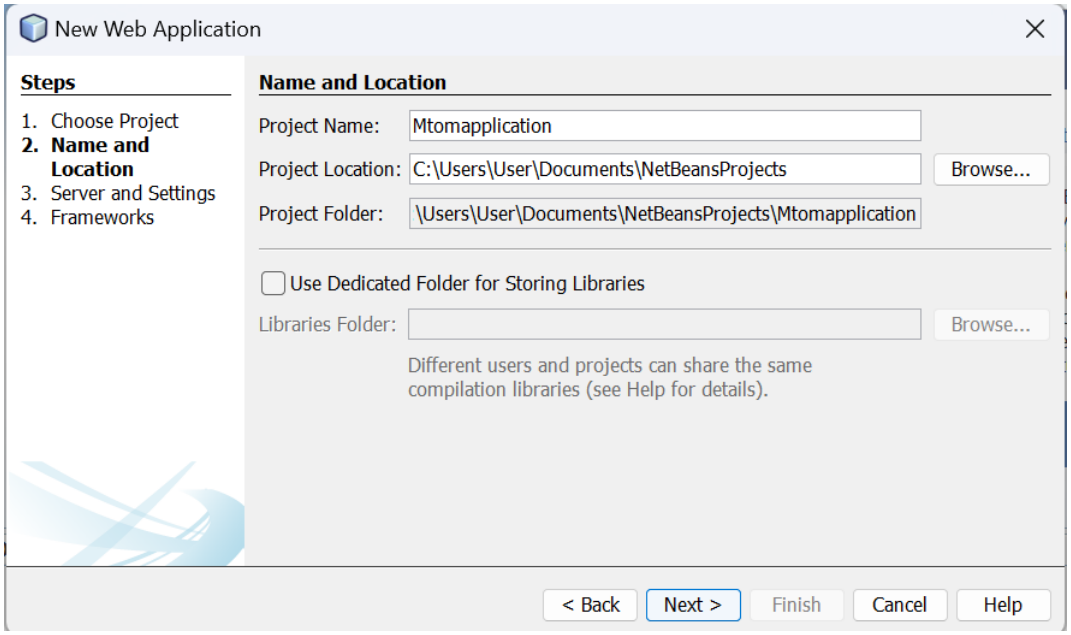


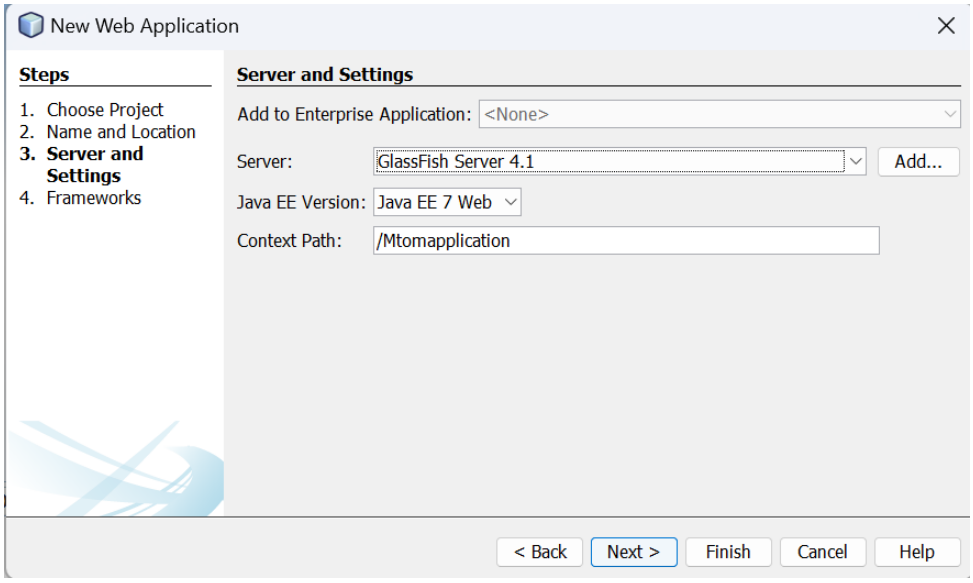


Select Java Web >Web Application

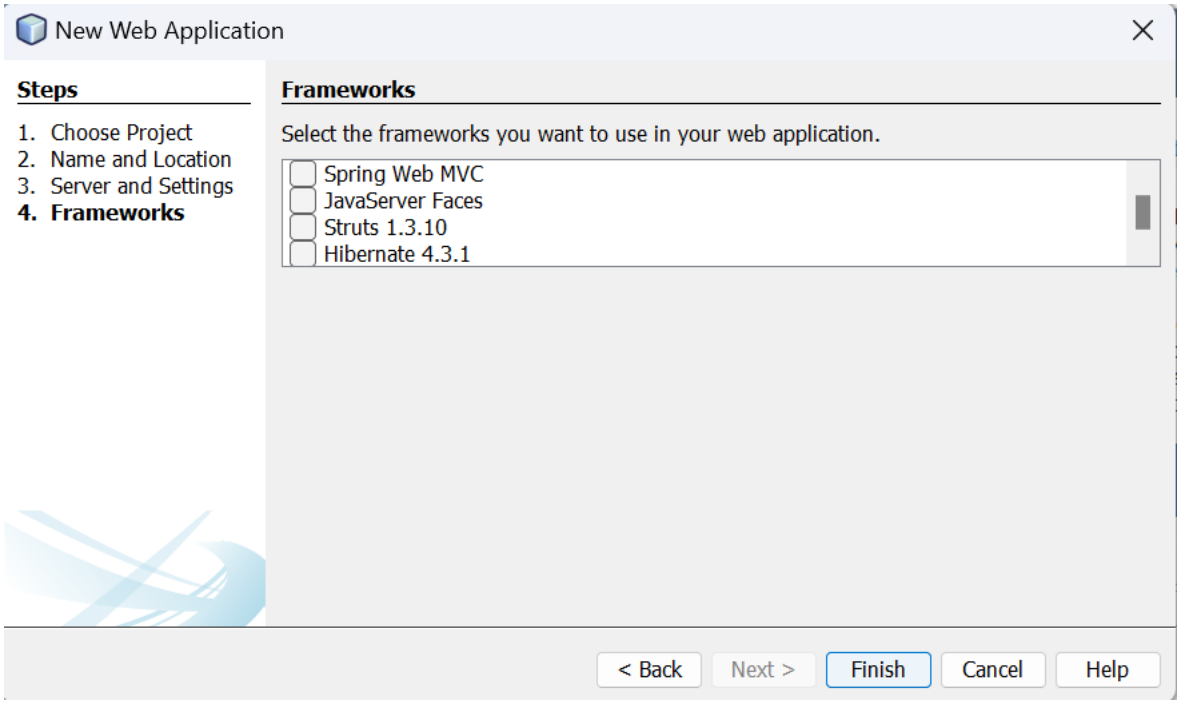

New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java
JavaFX
Java Web
Java EE
HTML5
Java ME Embedded
Java Card
Maven
Projects:
Web Application
Web Application with Existing Sou
Web Free-Form Application
Description:
Creates an empty Web application in a standard IDE project. A standard project uses an IDE-generated build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help

Click Next


New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Name and Location
Project Name: Mtomapplication
Project Location: C:\Users\User\Documents\NetBeansProjects
Browse...
Project Folder: \Users\User\Documents\NetBeansProjects\Mtomapplication
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
< Back
Next >
Finish
Cancel
Help

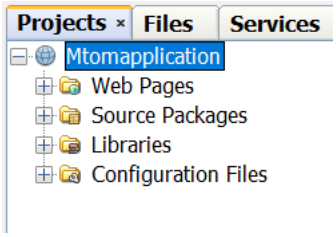


Click Next

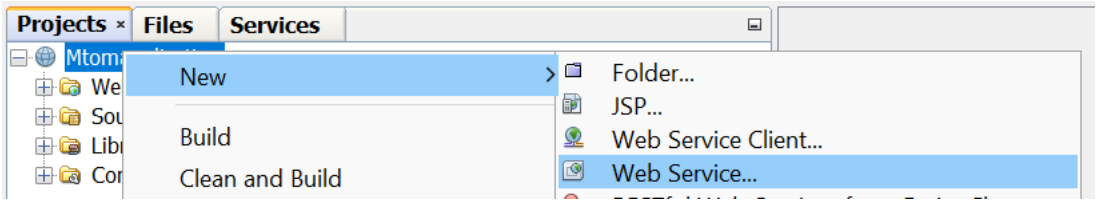


Click Finish

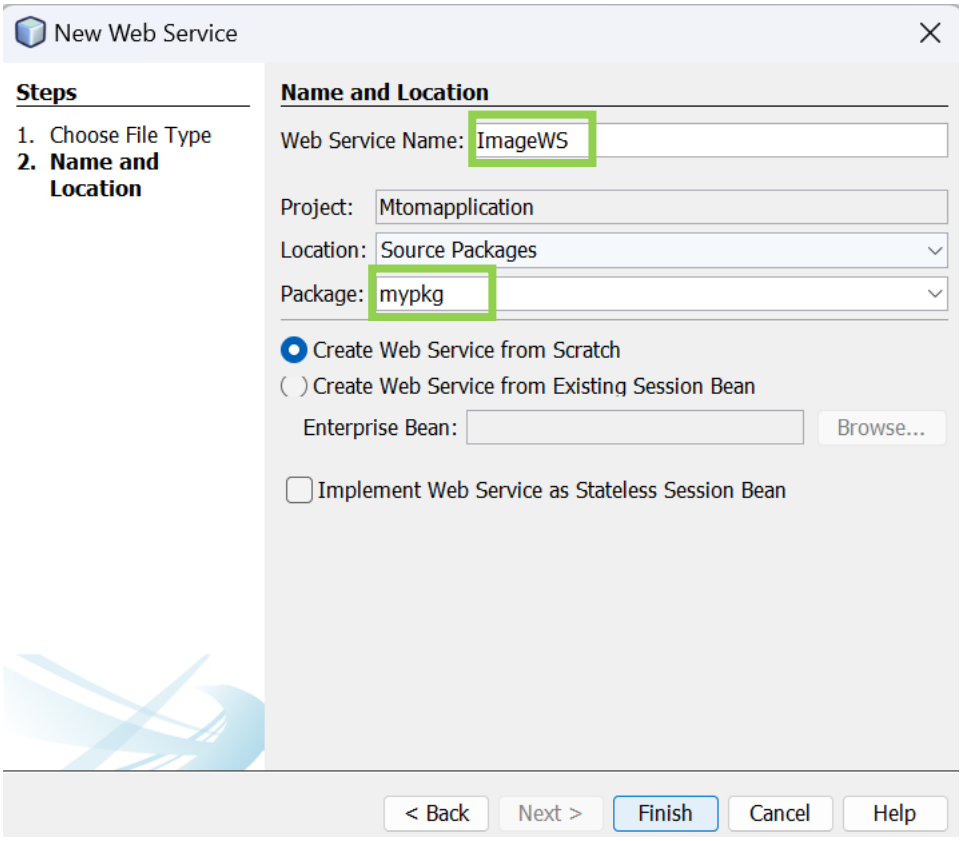
Expand Project Architecture



Right Click on Mtomapplication Project New > Web Service



Name: ImageWS and Package: mypkg and Finish



Filename: ImageWS.java

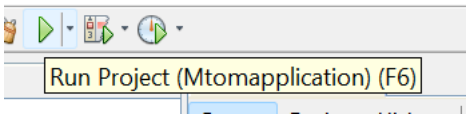
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
```

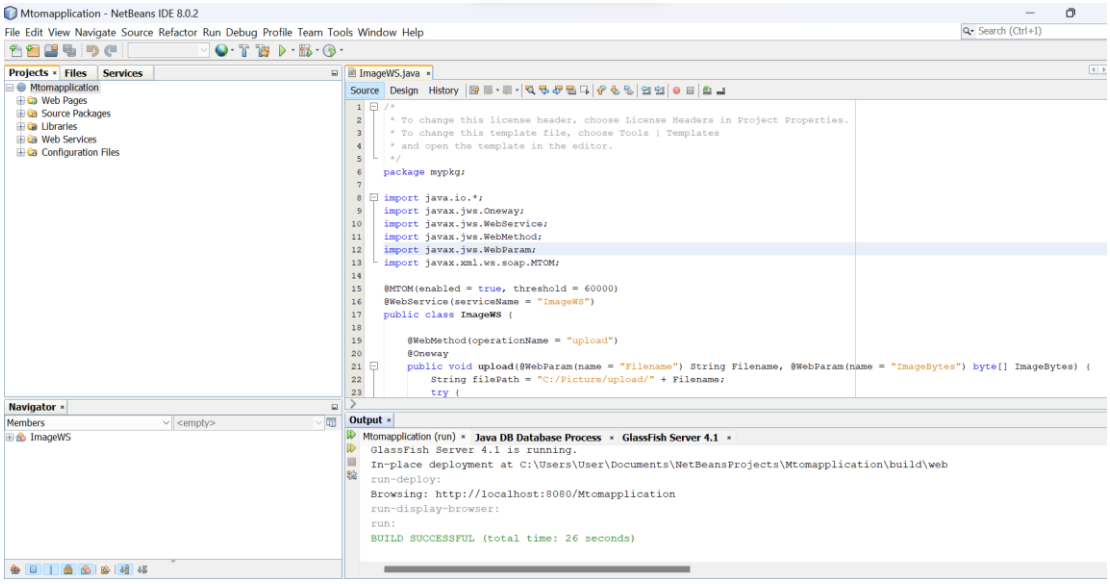
```
 */
package mypkg;


import java.io.*;
import javax.jws.Oneway;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.xml.ws.soap.MTOM;


@MTOM(enabled = true, threshold = 60000)
@WebService(serviceName = "ImageWS")
public class ImageWS {


    @WebMethod(operationName = "upload")
    @Oneway
    public void upload(@WebParam(name = "Filename") String Filename,
@WebParam(name = "ImageBytes") byte[] ImageBytes) {
        String filePath = "C:/Picture/upload/" + Filename;
        try {
            FileOutputStream fos = new FileOutputStream(filePath);
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            bos.write(ImageBytes);
            bos.close();
            System.out.println("Received file: " + filePath);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

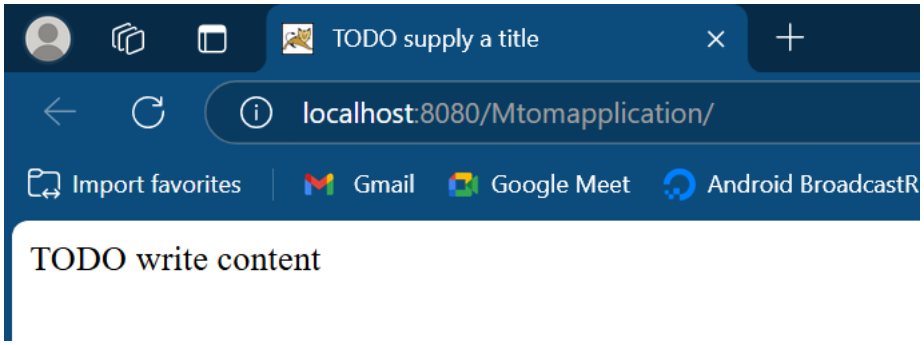
```
    @WebMethod(operationName = "download")
    public byte[] download(@WebParam(name = "Filename") String Filename) {
        String filePath = "C:/Picture/upload/" + Filename;
        System.out.println("Sending file: " + filePath);
        try {
            File file = new File(filePath);
            FileInputStream fis = new FileInputStream(file);
            BufferedInputStream bis = new BufferedInputStream(fis);
            byte[] fileBytes = new byte[(int) file.length()];
            bis.read(fileBytes);
            bis.close();
            return fileBytes;
        } catch (Exception ex) {
            ex.printStackTrace();
            return null;
        }
    }
}
```

Save File.

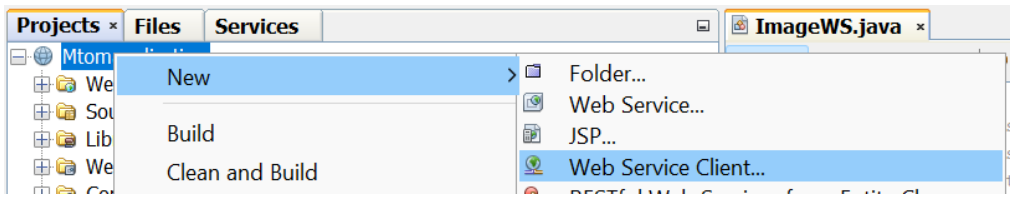
Run Project



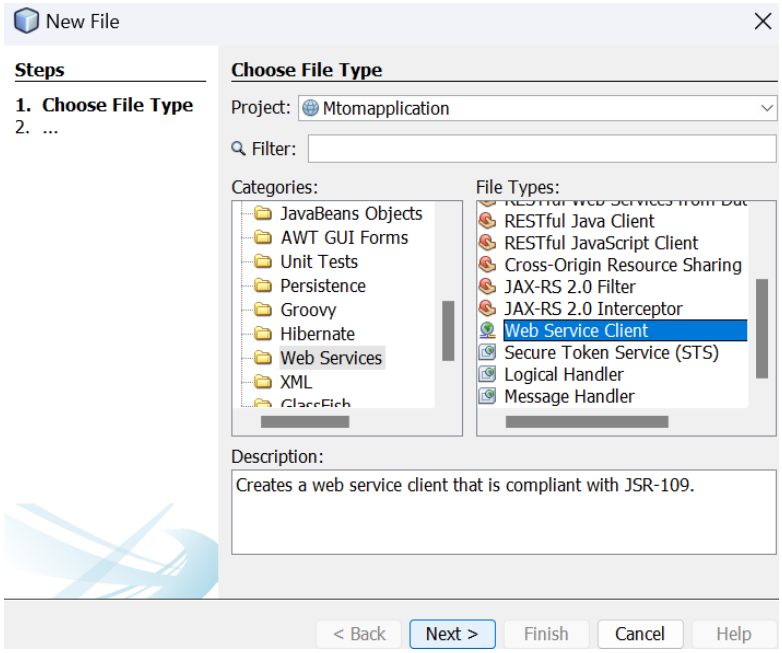




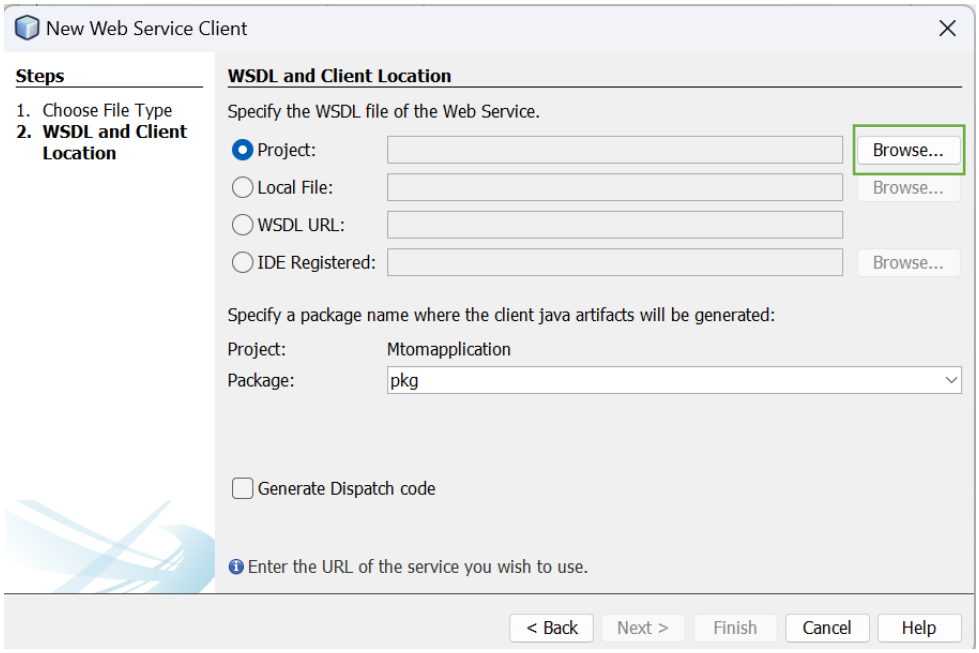
Right Click on Mtomapplication > New > Web Service Client.

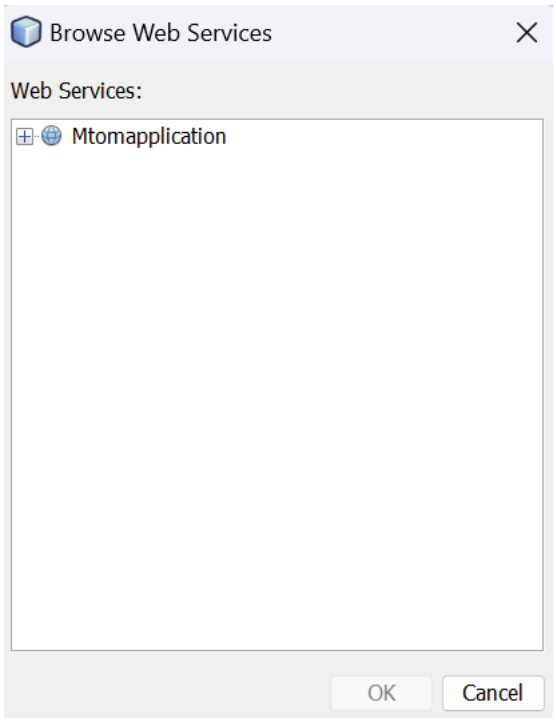


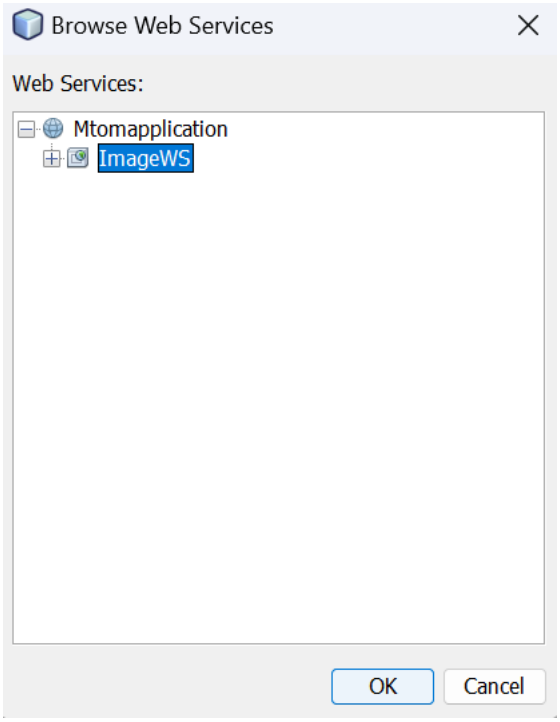
If not available click on Others



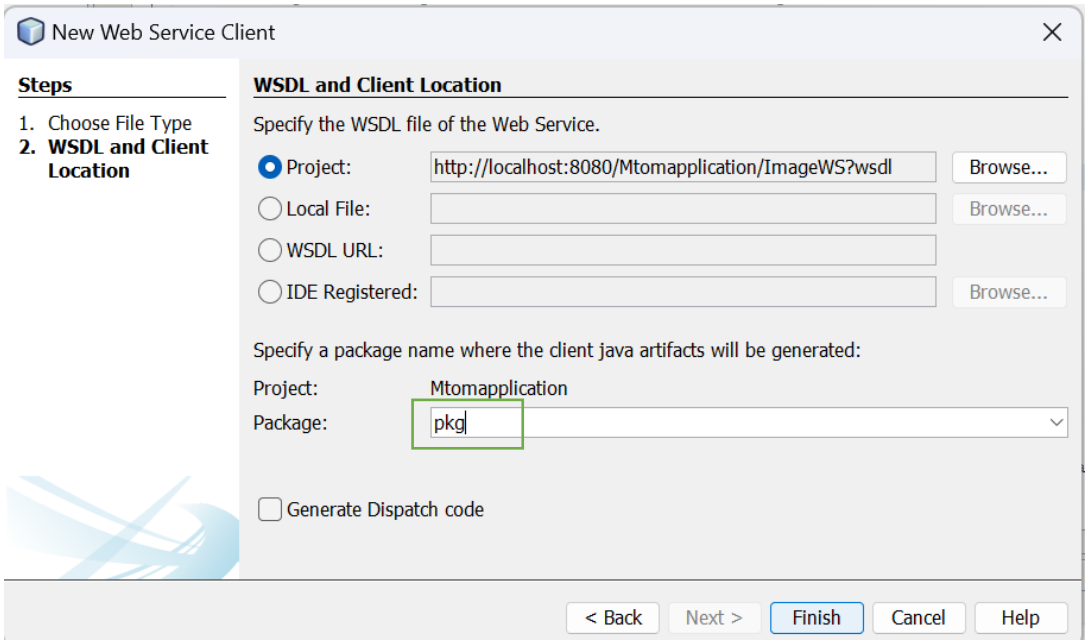
Select on Browse


New Web Service Client
Steps
1. Choose File Type
2. WSDL and Client Location
WSDL and Client Location
Specify the WSDL file of the Web Service.
Project:
Local File:
WSDL URL:
IDE Registered:
Browse...
Browse...
Browse...
Specify a package name where the client java artifacts will be generated:
Project: Mtomapplication
Package: pkg
Generate Dispatch code
Enter the URL of the service you wish to use.
< Back
Next >
Finish
Cancel
Help


Browse Web Services
Web Services:
Mtomapplication
OK
Cancel

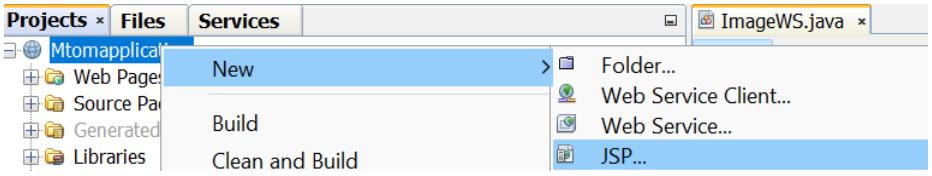

Browse Web Services
Web Services:
Mtomapplication
ImageWS
OK
Cancel

Package: pkg

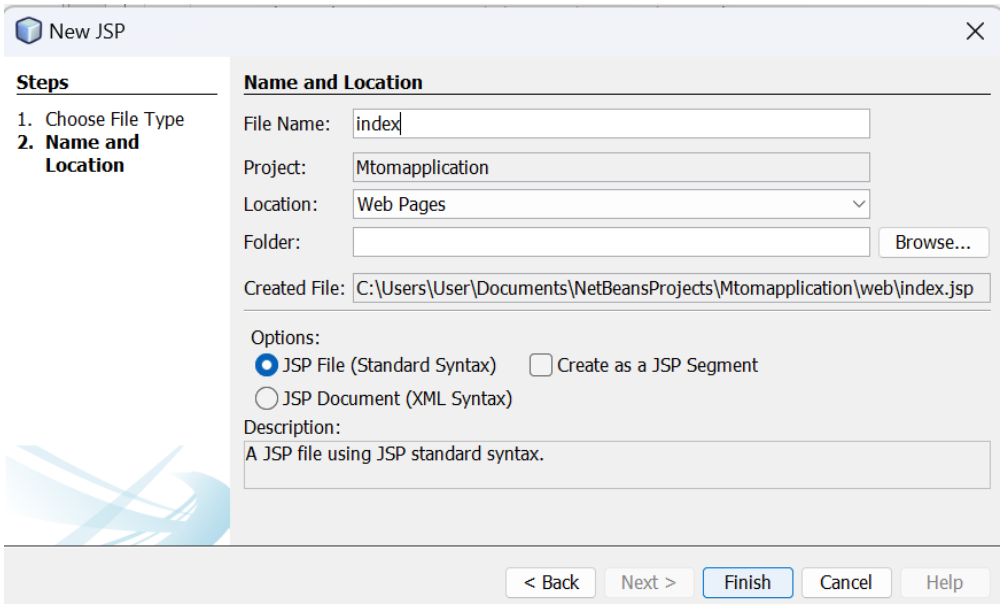


Finish

New > JSP



Name: index and click Finish



Filename: index.jsp

<%--

```
    Document   : index
    Created on : Feb 3, 2025, 1:25:10 PM
    Author     : User
--%>
<%@page import="java.io.BufferedOutputStream"%>
<%@page import="java.io.FileOutputStream"%>
<%@page import="java.io.FileInputStream"%>
<%@page import="java.io.BufferedInputStream"%>
<%@page import="java.io.File"%>
<%@page import="javax.xml.ws.soap.MTOMFeature"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
</head>
<body>
<%-- start web service invocation --%><hr/>
<%
try {
    pkg.ImageWS_Service service = new pkg.ImageWS_Service();
    pkg.ImageWS port = service.getImageWSPort(new MTOMFeature(60000));



    // TODO initialize WS operation arguments here
    String filePath="C:/Picture/abcd2.jpg"; // Assuming this is the correct file path
    File file=new File(filePath);
    FileInputStream fis=new FileInputStream(file);
    BufferedInputStream bis=new BufferedInputStream(fis);
```

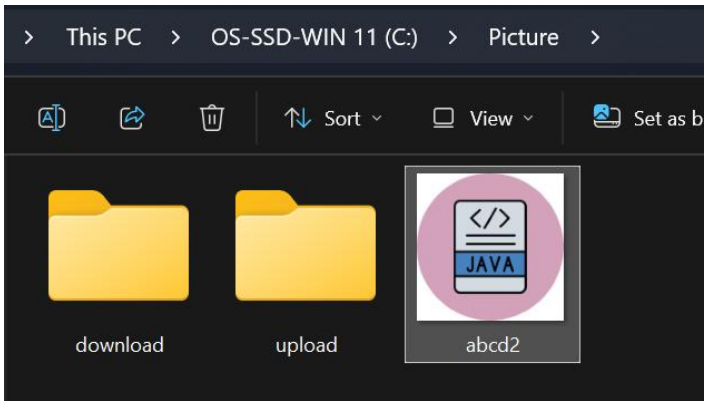
```
    String filename = file.getName();
    byte[]imageBytes=new byte[(int)file.length()];
    bis.read(imageBytes);
        port.upload(filename, imageBytes);
    bis.close();
        out.println("File uploaded :" + filePath);
} catch (Exception ex) {
    // TODO handle custom exceptions here
    ex.printStackTrace();
}
%>
<%-- end web service invocation --%><hr/>
<%-- start web service invocation --%><hr/>
<%
try {
    pkg.ImageWS_Service service = new pkg.ImageWS_Service();
    pkg.ImageWS port = service.getImageWSPort();
        // TODO initialize WS operation arguments here
    String filename = "abcd2.jpg"; // Assuming this is the correct file name




    // Initialize filePath
    String filePath = "C:/Picture/download/" + filename; // Assuming this is the correct
download path
        // Invoke the download method and get the file bytes
    byte[] fileBytes = port.download(filename);
        // Write the downloaded bytes to a file
    FileOutputStream fos = new FileOutputStream(filePath);
```

```
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    bos.write(fileBytes);
    bos.close();
        out.println("File downloaded: " + filePath);
} catch (Exception ex) {
    // TODO handle custom exceptions here
    ex.printStackTrace();
}
%>
<%-- end web service invocation --%><hr/>
</body>
</html>
```

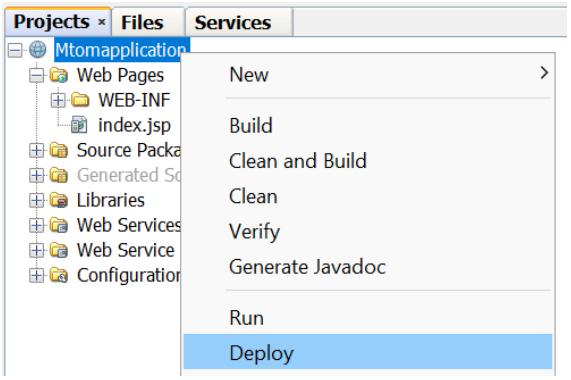
Create a Folder Name: Picture inside

Create 2 Folders name: download and upload and an image with name abcd2 or any name with proper extension

Here image name and extension: abcd2.jpg



Deploy Project



Then Run Project or F6