

**Hindi Vidya Prachar Samiti's RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE & COMMERCE (EMPOWERED AUTONOMOUS)**

**M.Sc. Computer Science Semester-I**

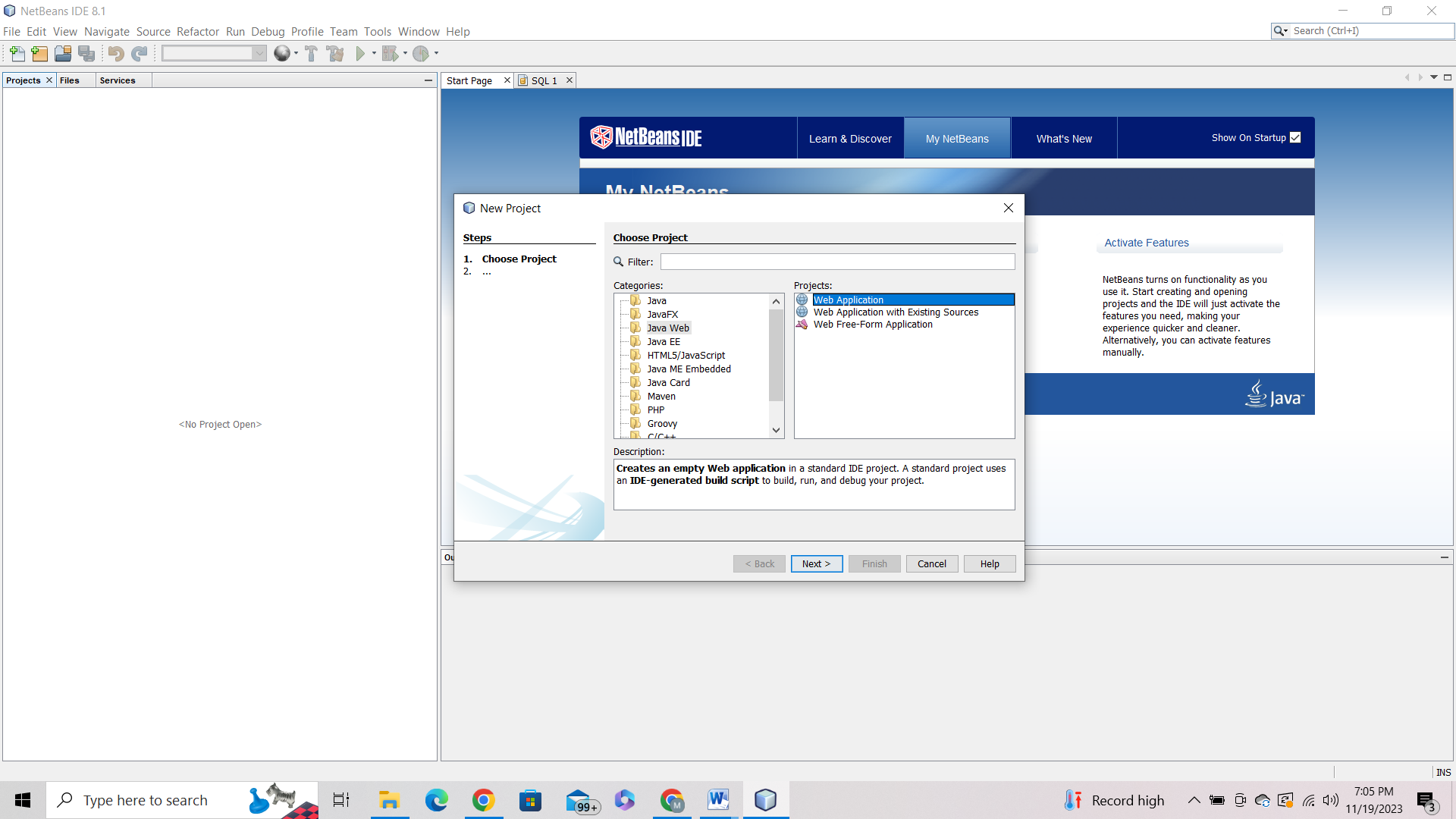
**Name:** **MEGHANA YASHWANT NARVEKAR** **Roll No**: **536**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX** | | | | |
| **NO** | **DATE** | **TITLE** | **PAGE NO** | **SIGN** |
|  | | | | |
| 1 |  | SOAP Webservices | 02 |  |
| 2 |  | Create RestFul Services to perform CRUD operation | 06 |  |
| 3 |  | Create RestFul Services from Pattern | 13 |  |
| 4 |  | MVC application. | 19 |  |
| 5 |  | Service Endpoint | 29 |  |
| 6 |  | Restful services using WEB-API. | 35 |  |
| 7 |  | Web application using Azure | 51 |  |
| 8. |  | Install Google App Engine. Create hello world app and other simple web applications using python/java. | 57 |  |

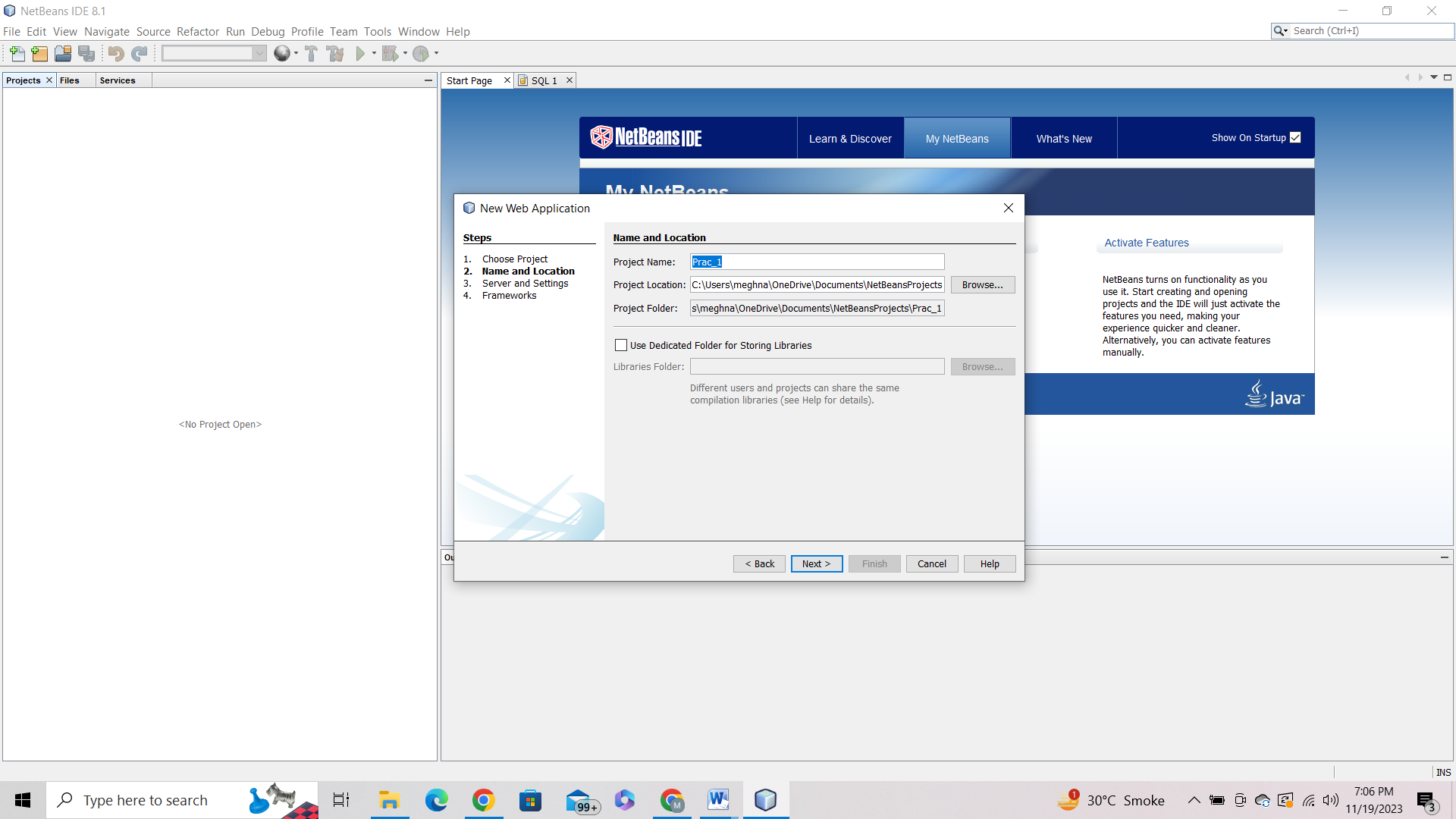
**Practical\_No : 01**

**Title : SOAP Webservices**

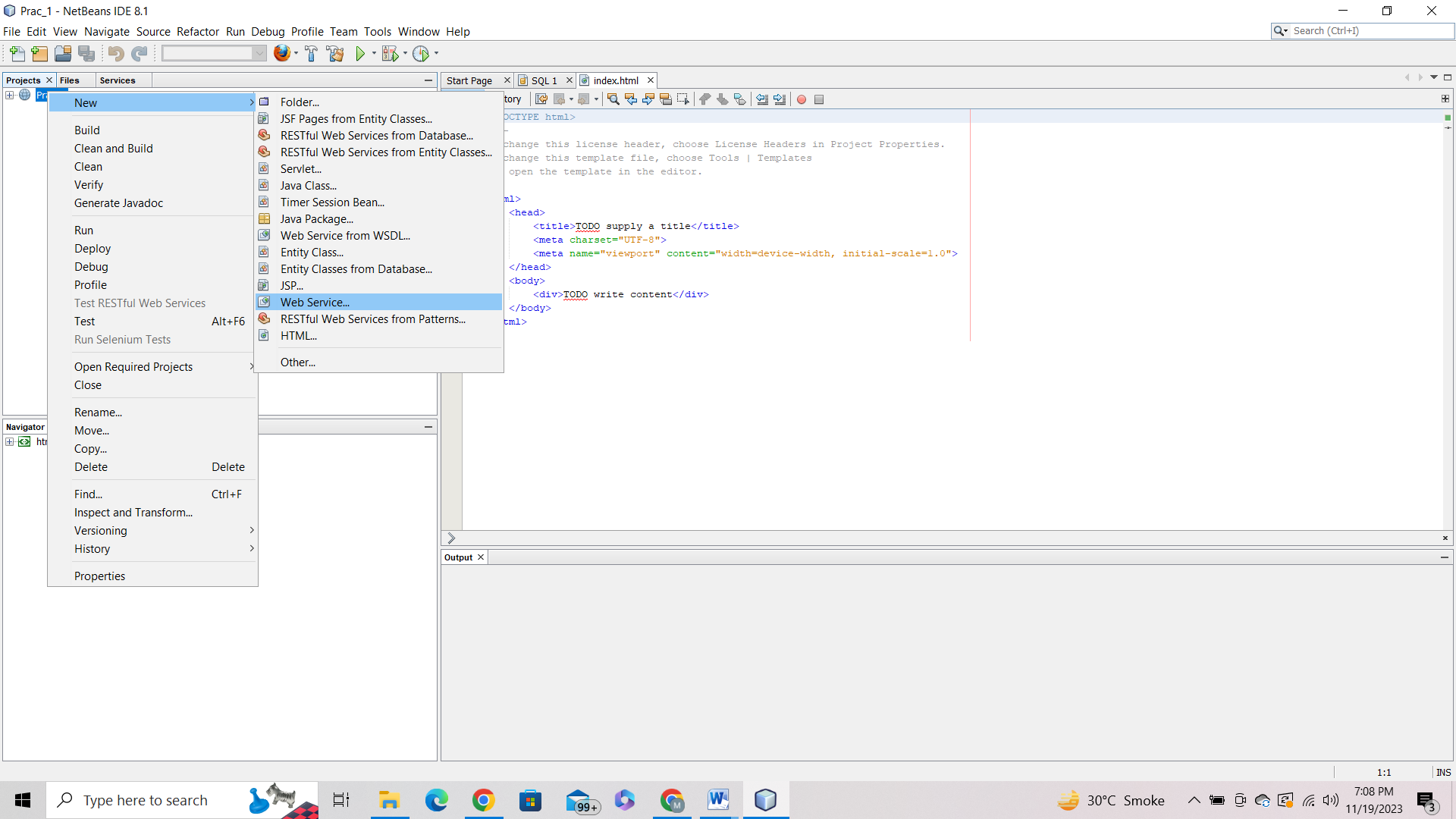
Open NetBeans > Open New Project > Select Java Web > Web Application



Give Project Name > Select GlassFish Server



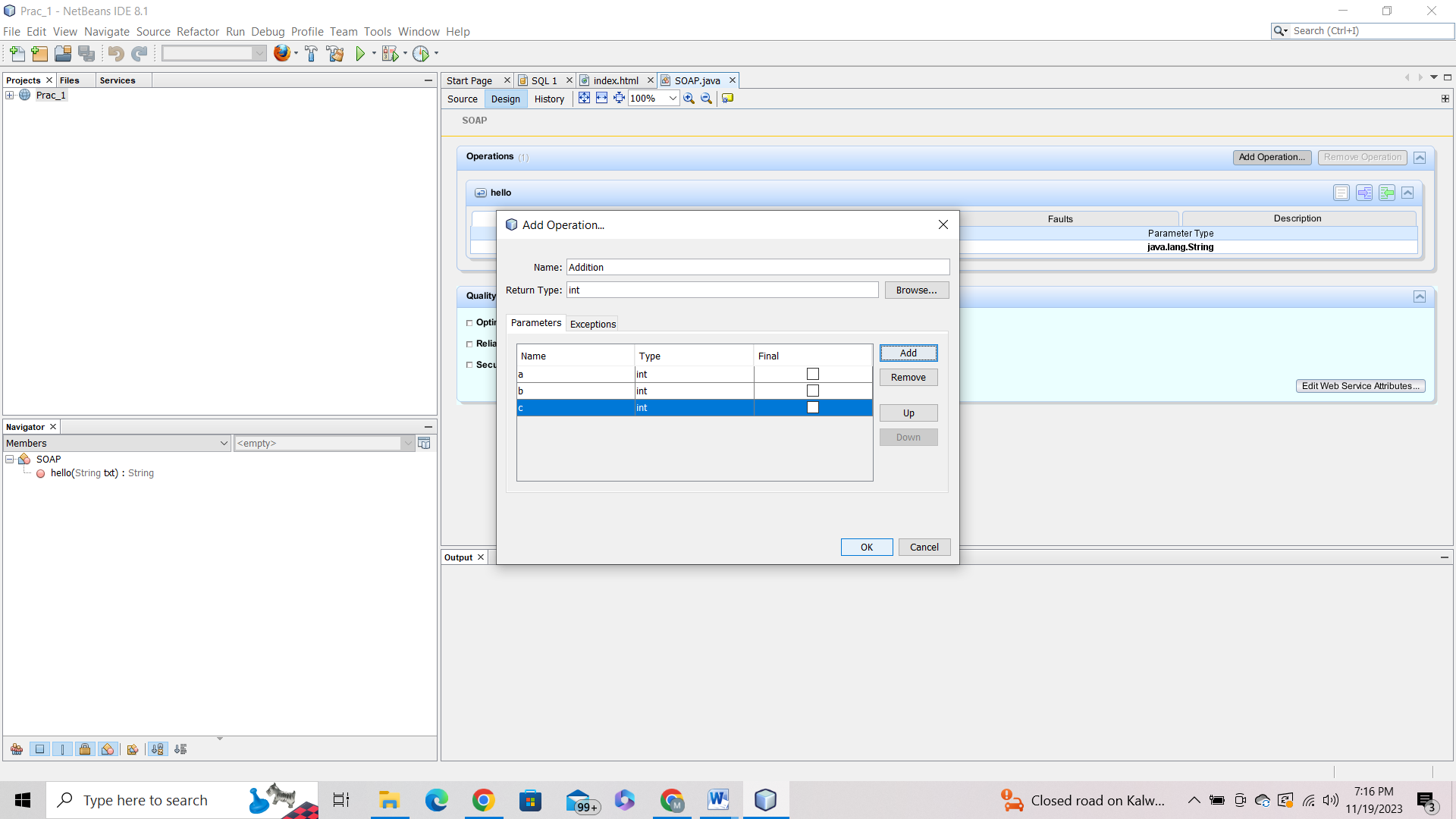
Right Click On Project > New > Select Web Service



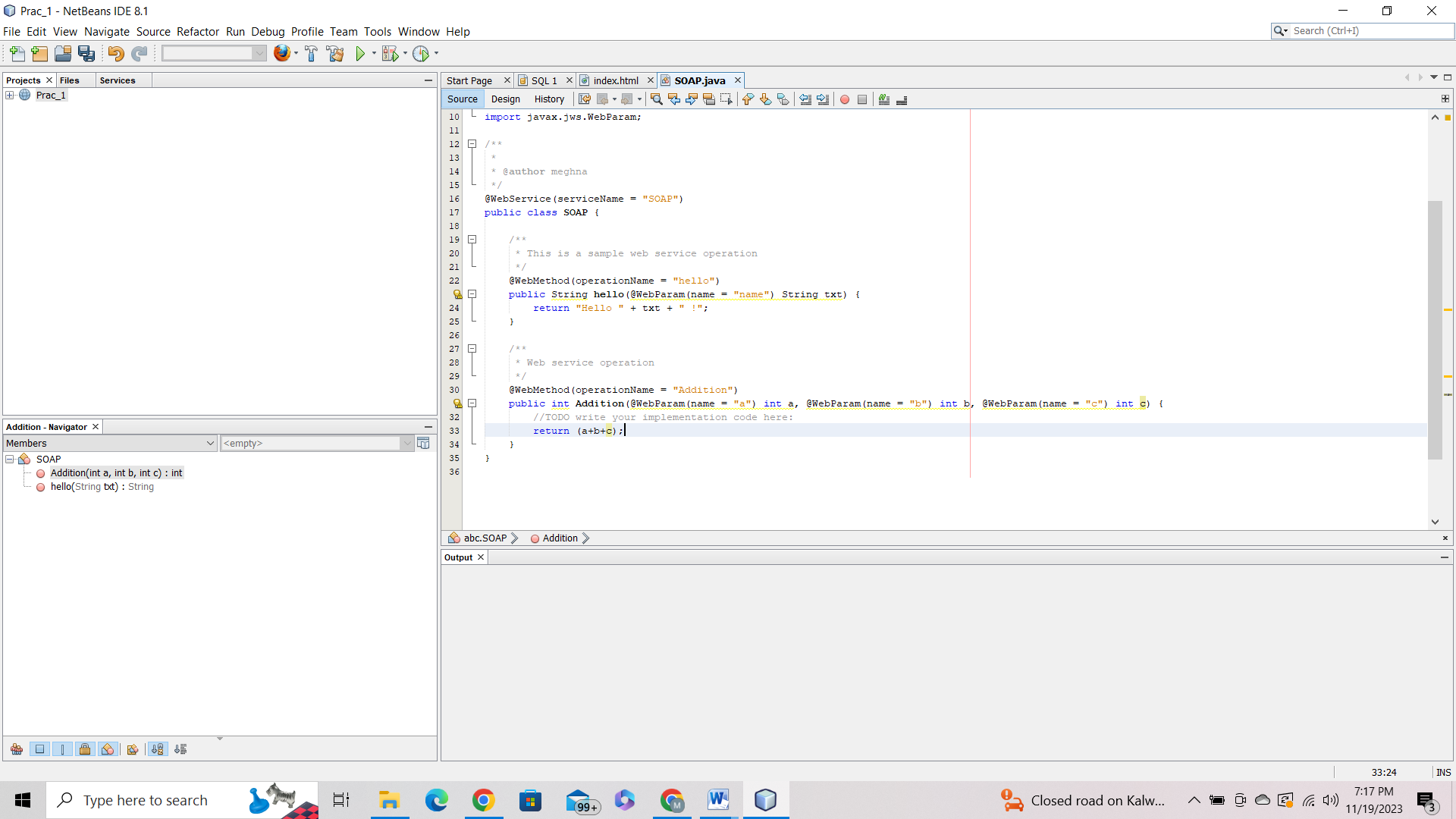
Give Web Service Name and Package Name > Finish



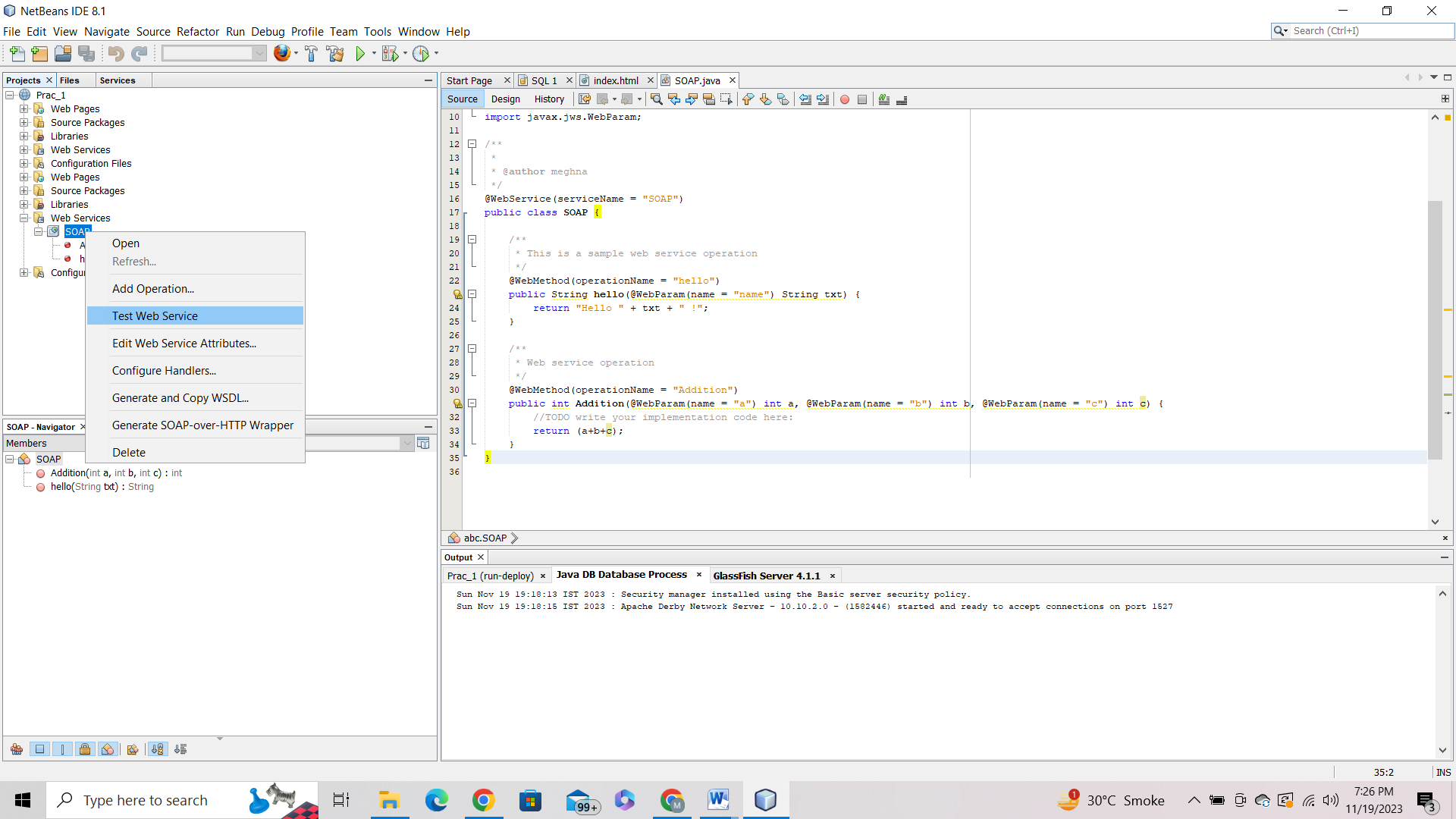
Go to design > Add Operation Name > Return type – int >Add Parameters



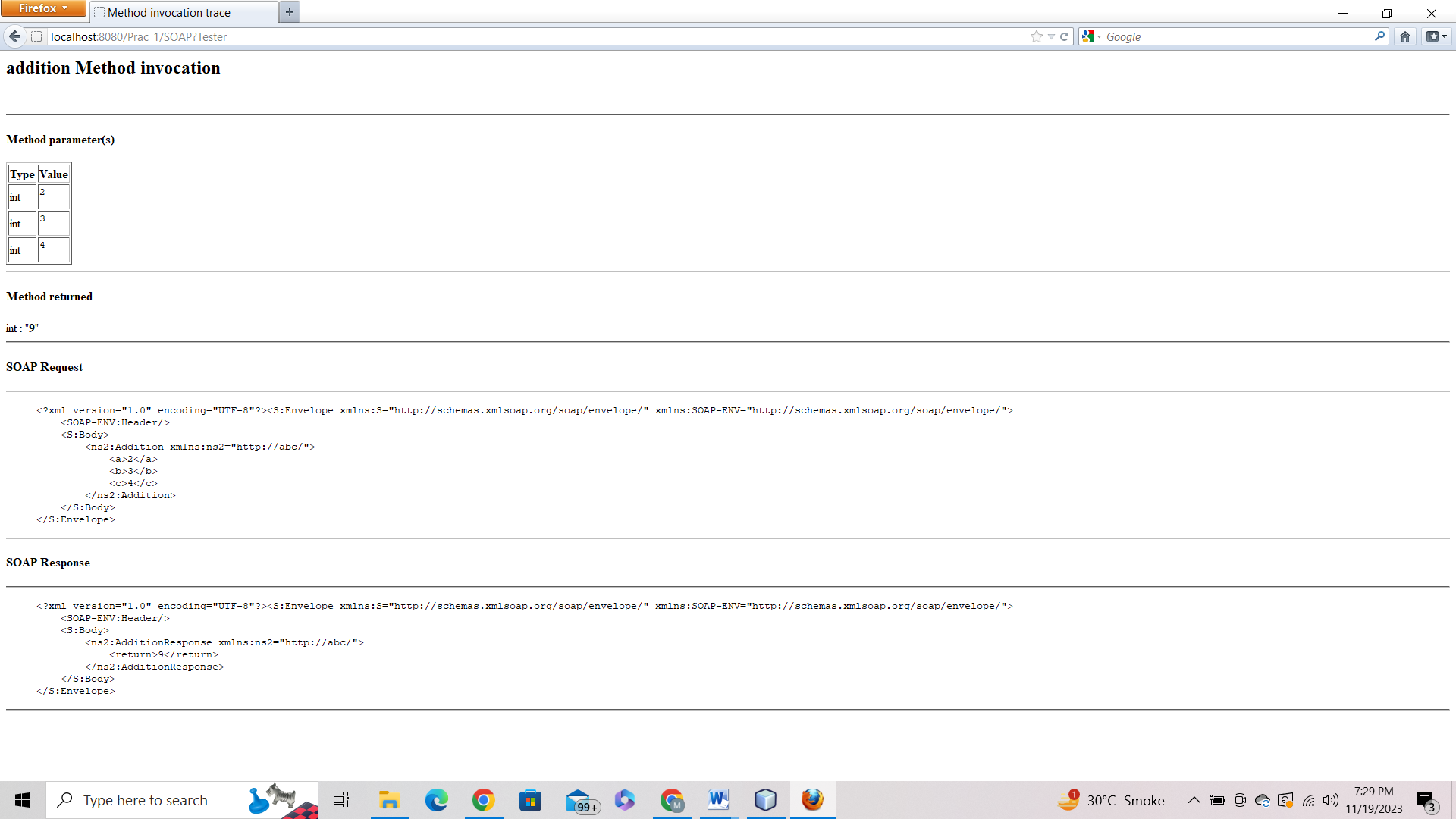
Write Operation in Return



Deploy the Project > Test the Web Service



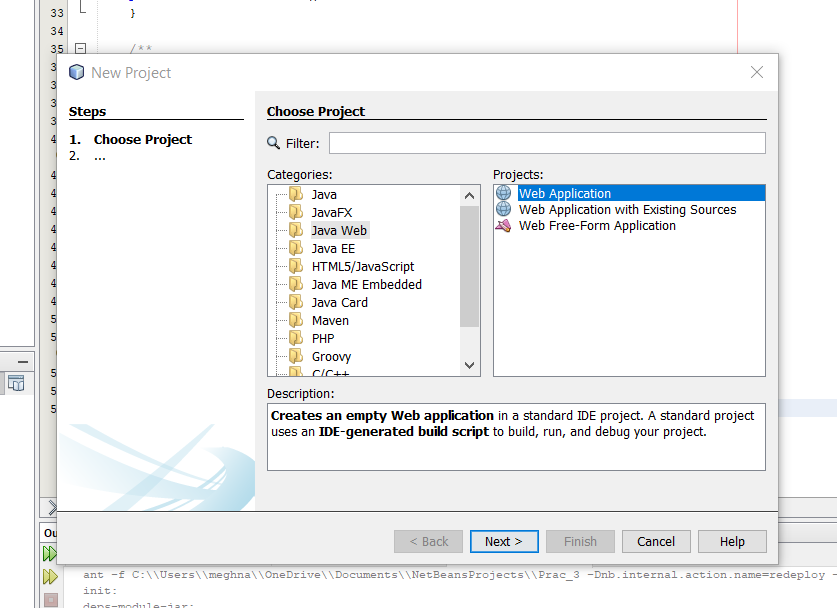
Give Values and click the operation name



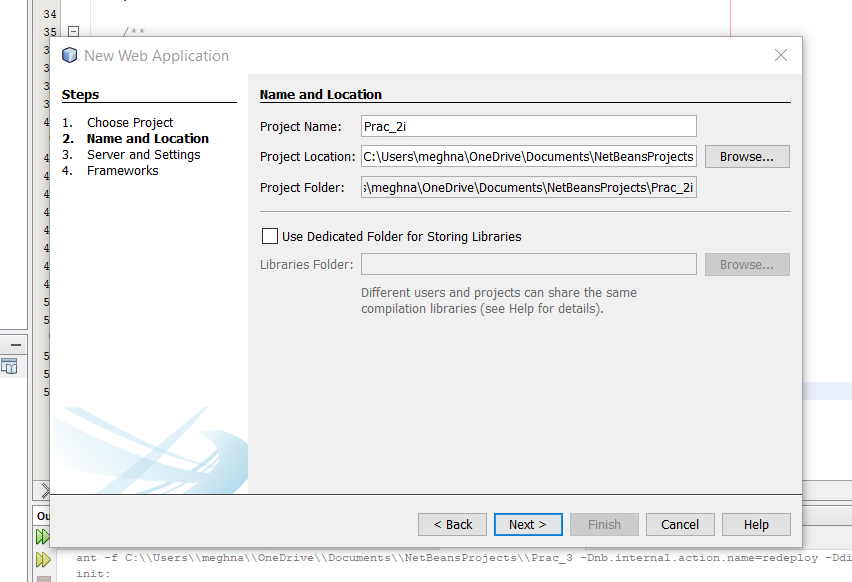
**Prac\_No : 02**

**Title : Create RestFul Services to Perform CRUD Operations**

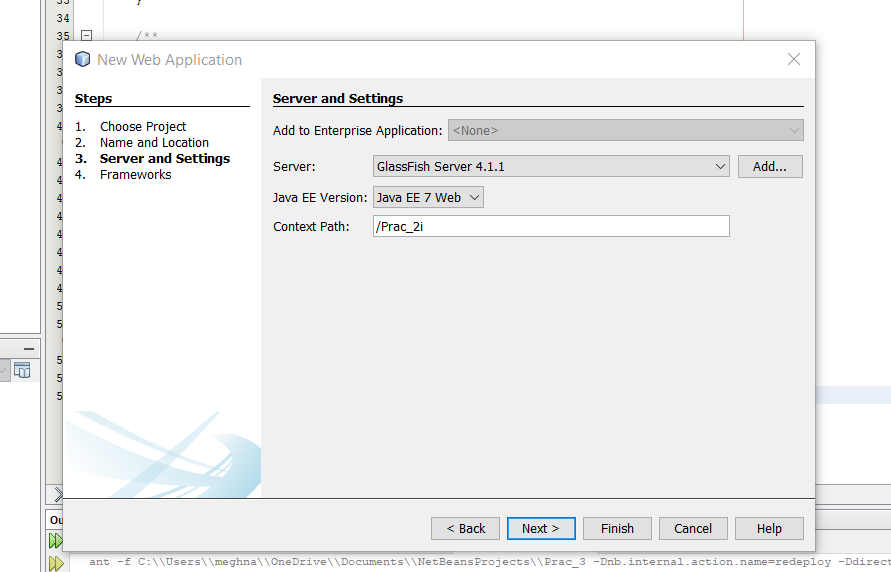
Open NetBeans > Create New Project > Select Java Web > Web Application



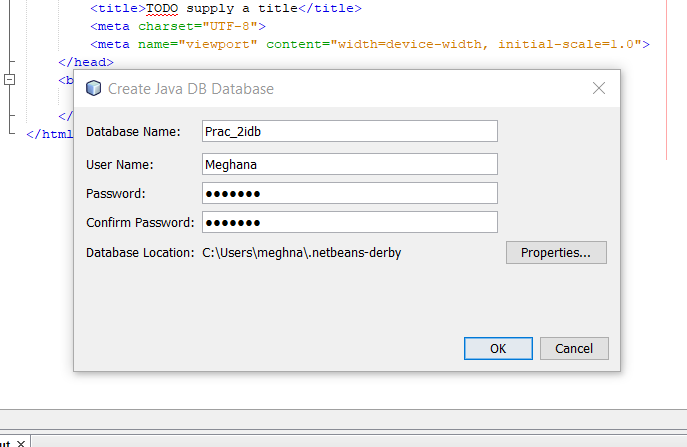
Give Project Name



Select GlassFish Server



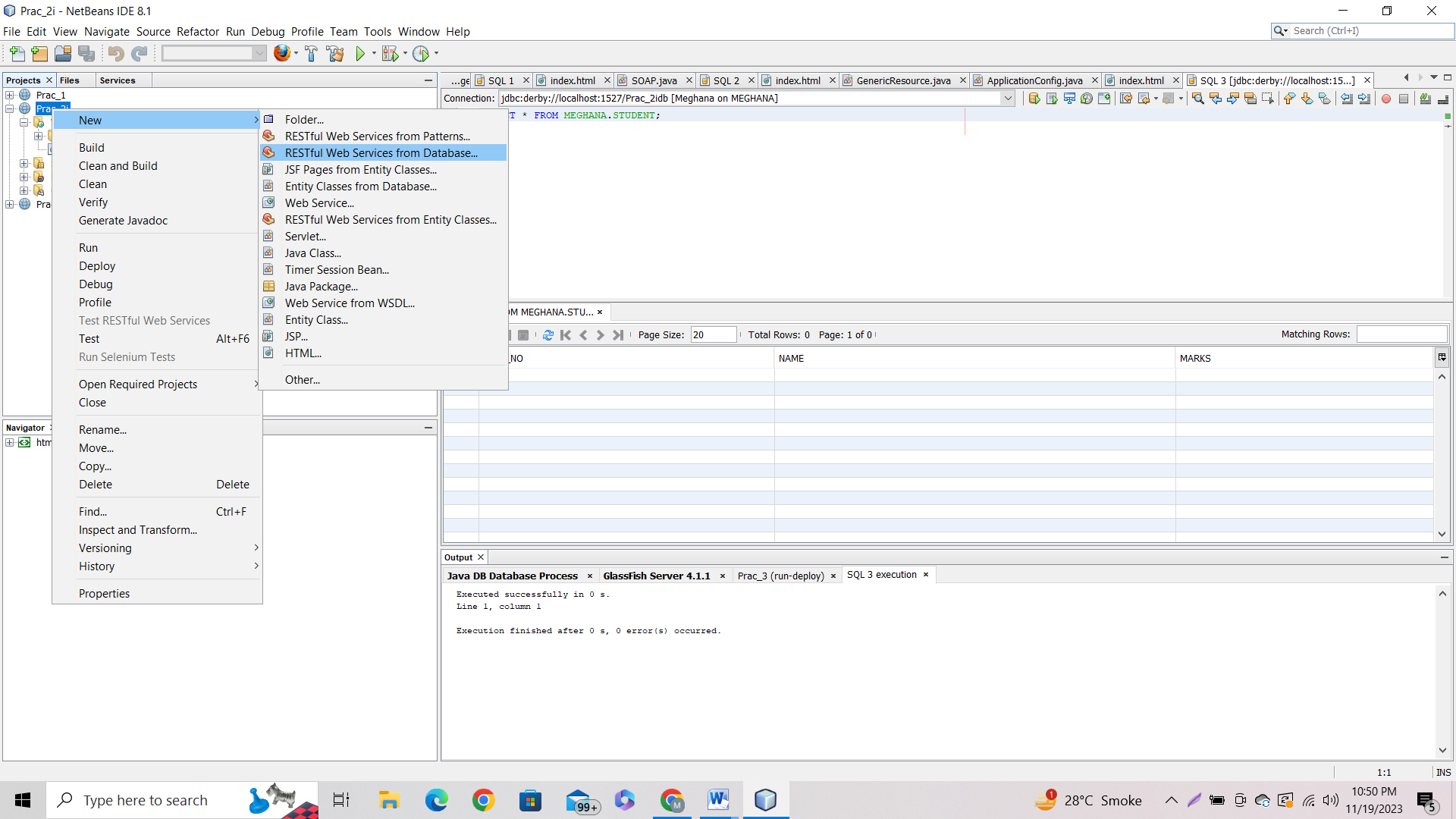
Go in Services > Right Click on JavaDB > Create New Database



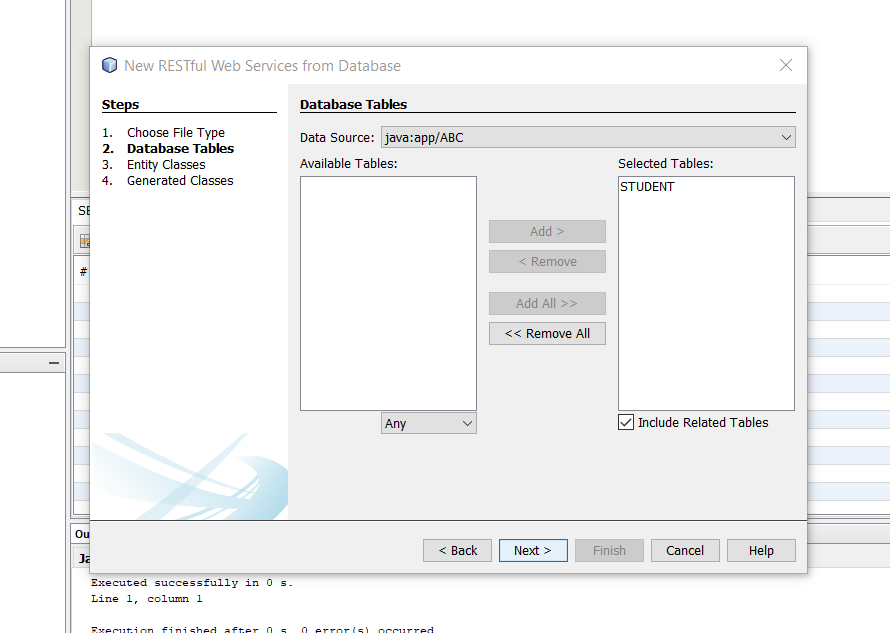
Connect Database connection > Create Table



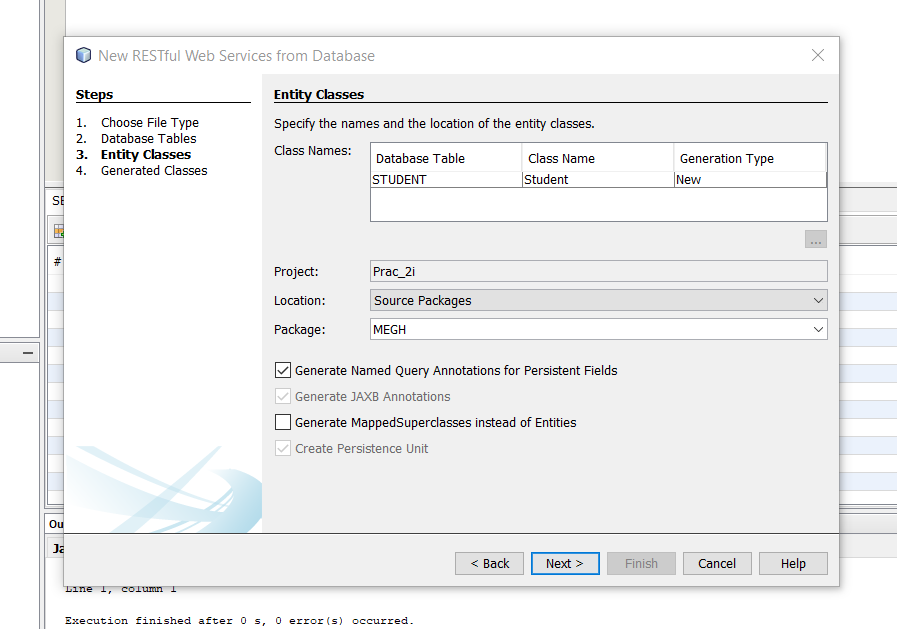
Right Click On Project > New > Select RestFul Web Services from Database



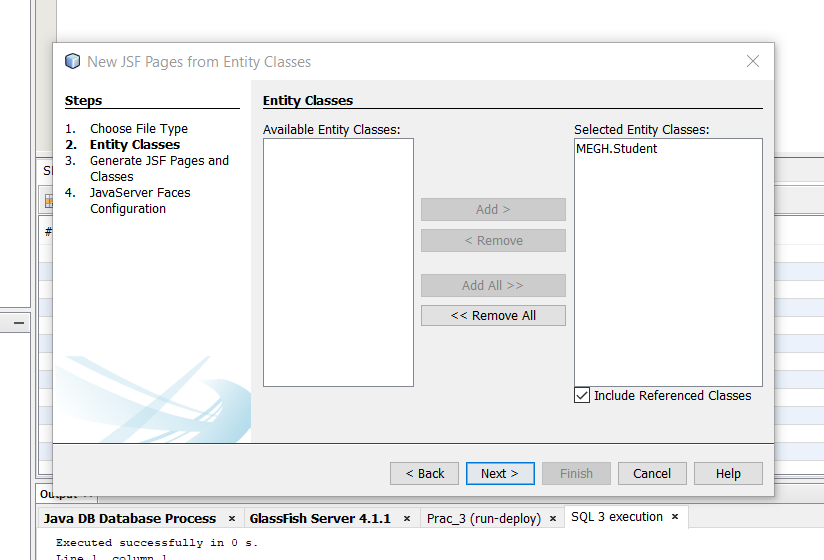
Add Data Source > Add table



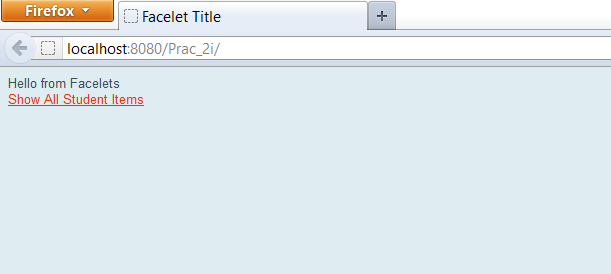
Give Package name

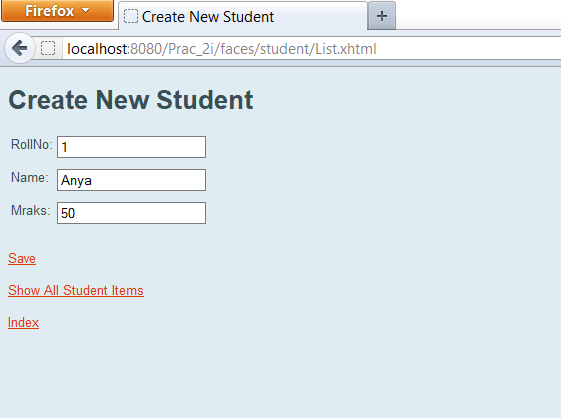


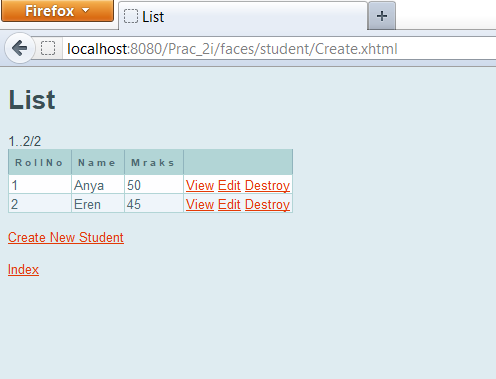
Right Click on Project > New > JSF Pages From Entity Classes > Select Entity Class



Output :



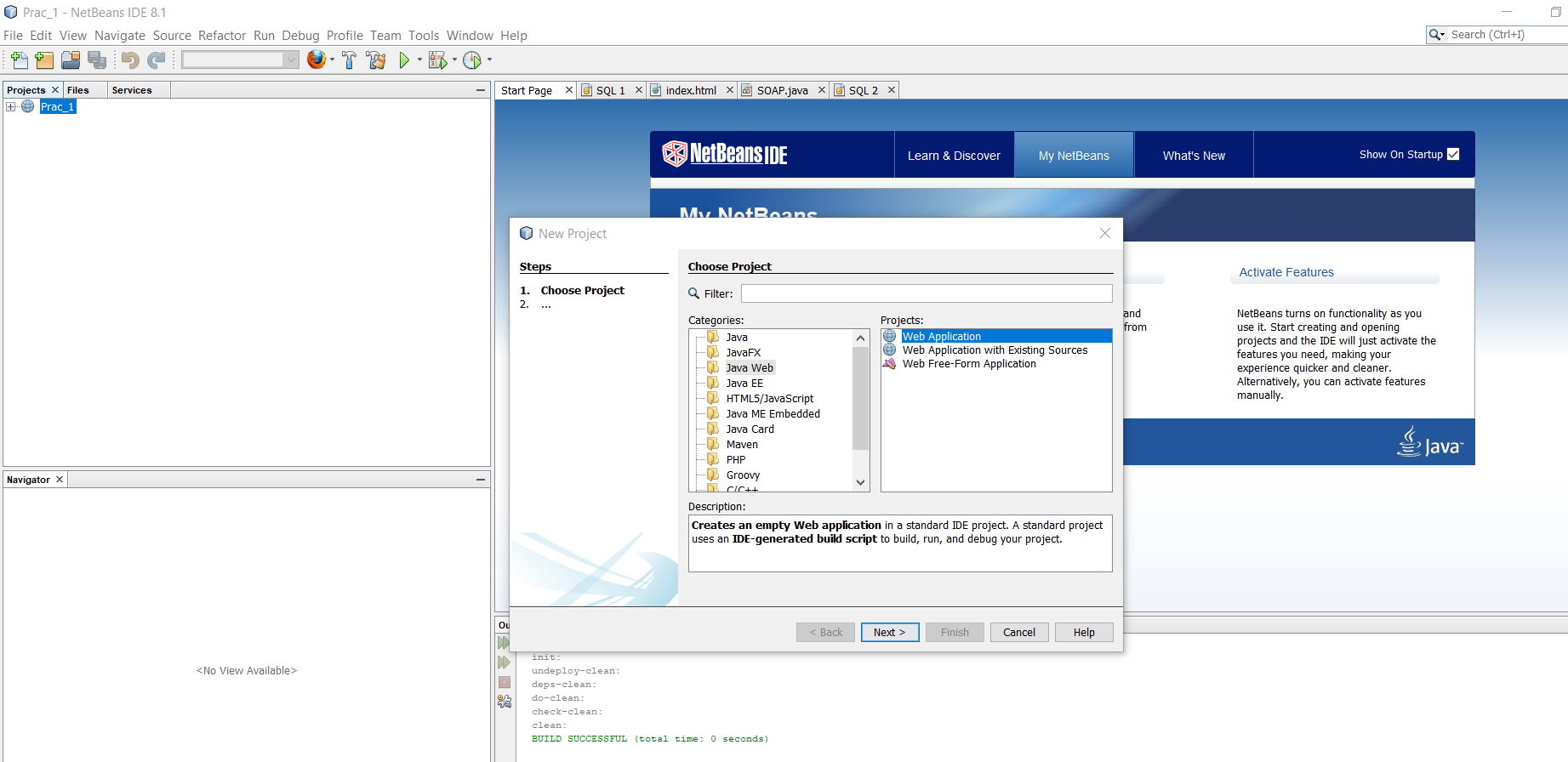




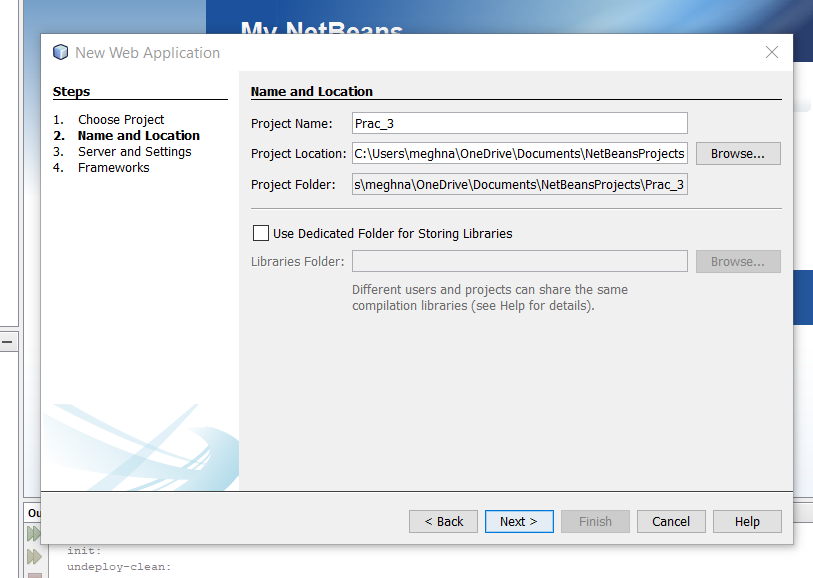
**Prac\_No : 03**

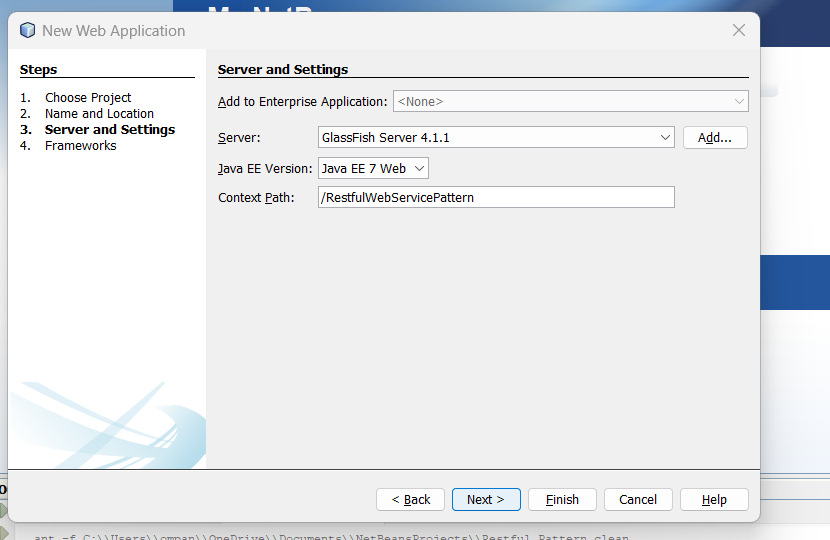
**Title : Create RestFul Services From Pattern**

Open NetBean > Create New Project > Select Java Web > Web Application

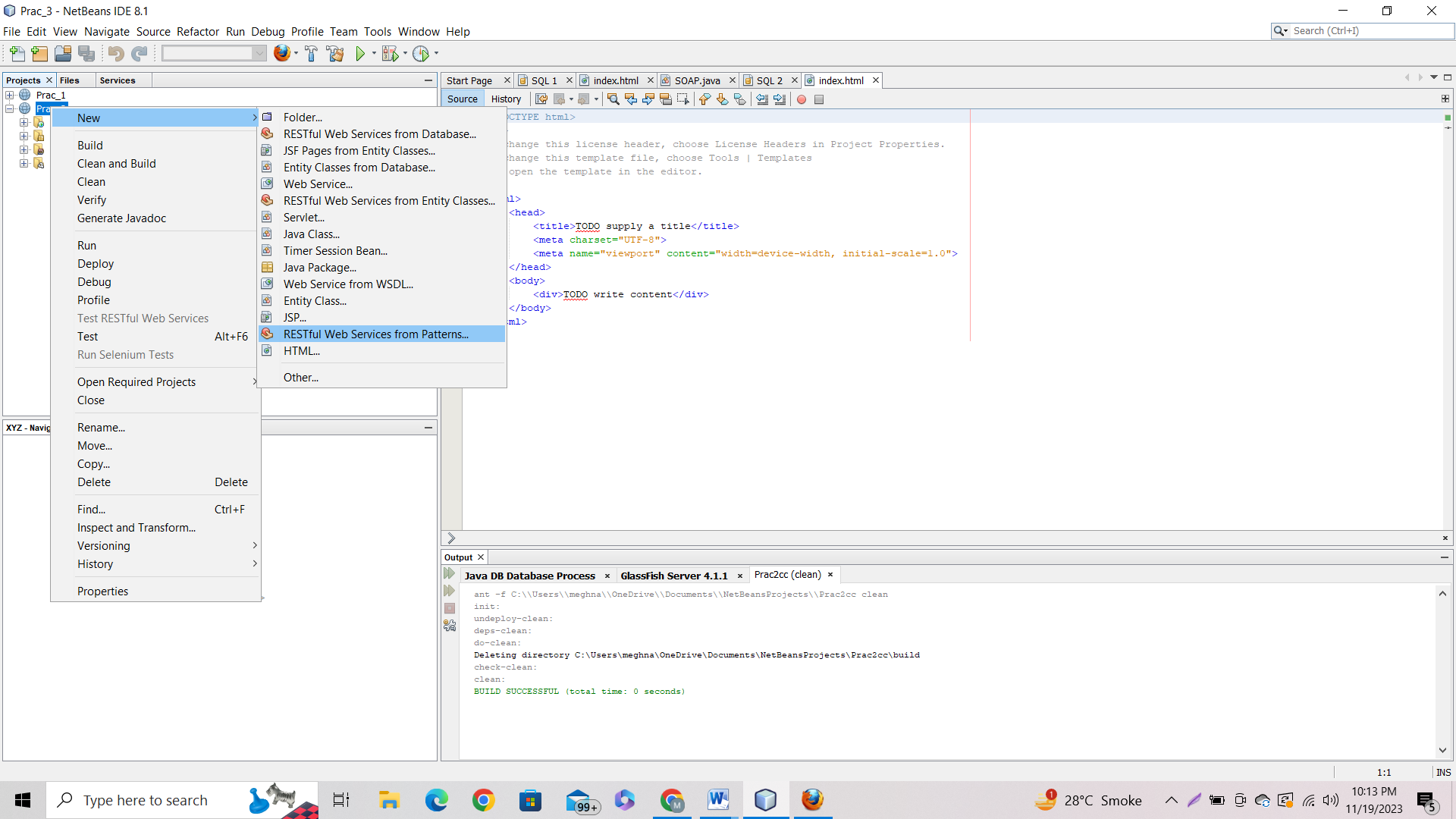


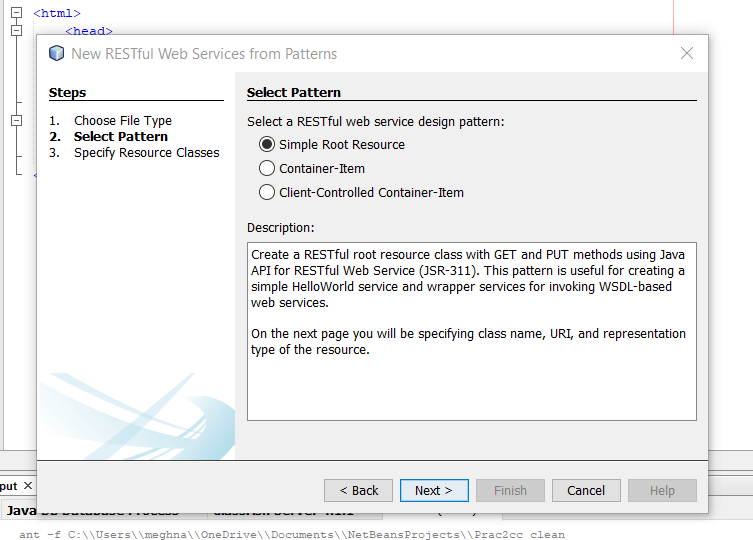
Give Project Name > Next > Select GlassFish Server



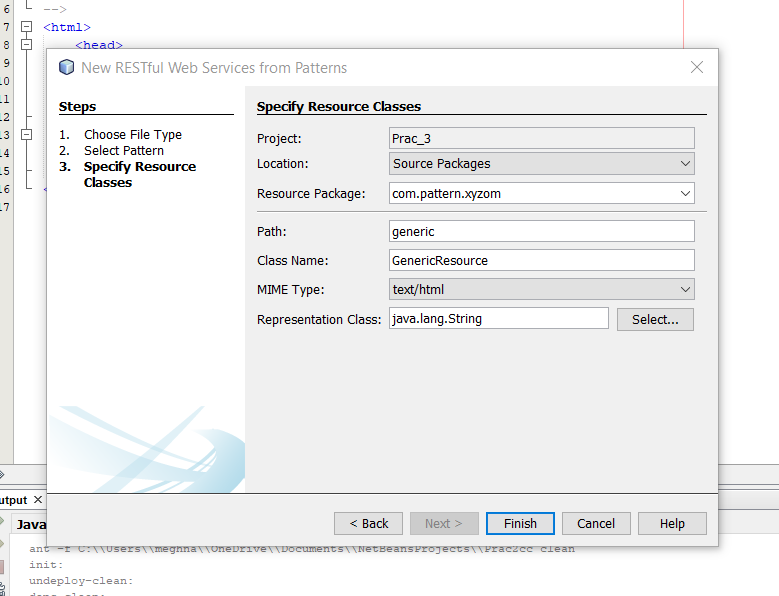


Right Click on Project > Select Restful web services from patterns



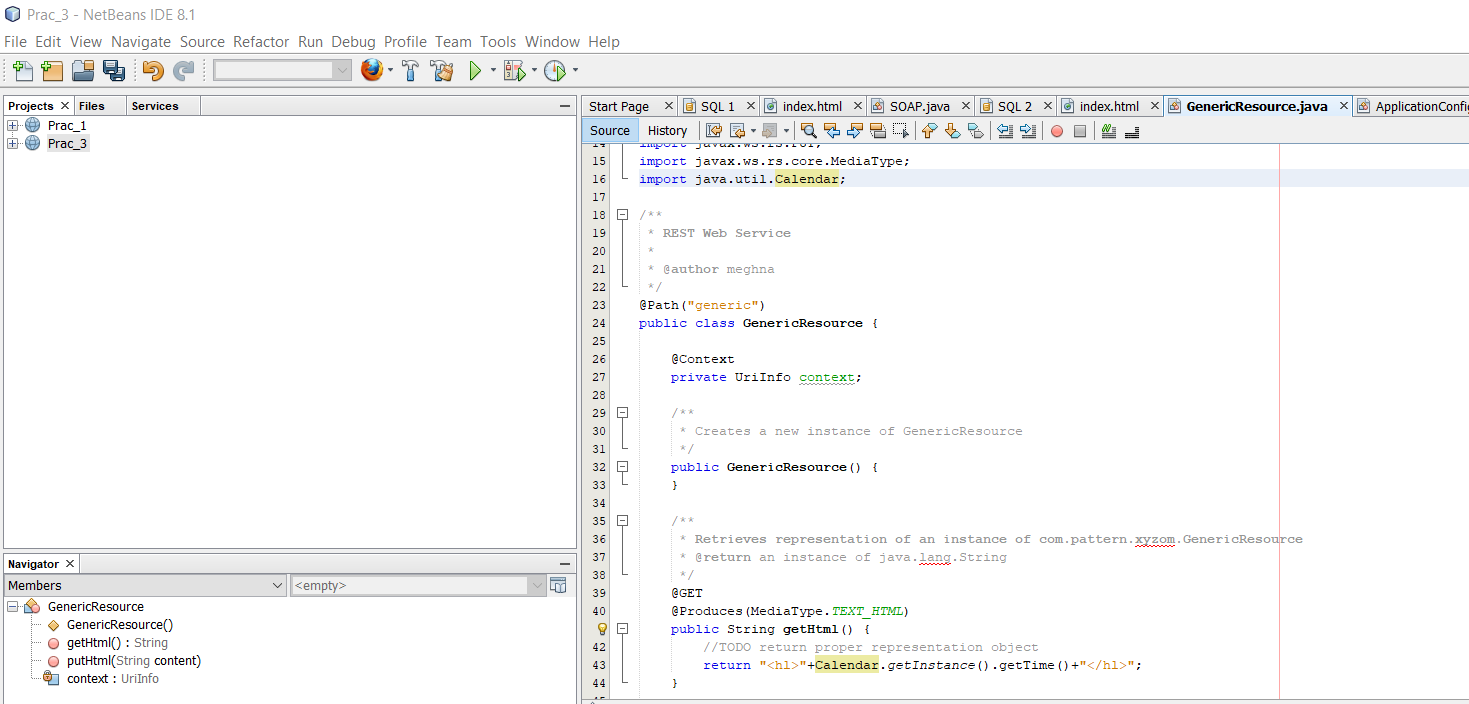


Give Resource Name and Change MIME Type into text/html



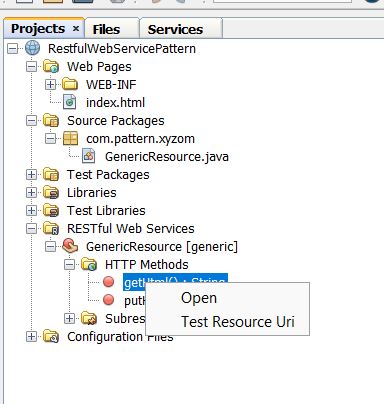
Add import java.util.Calendar;

Add Code : return “<hl>”+Calendar.getInstance().getTime()+”</hl>”;

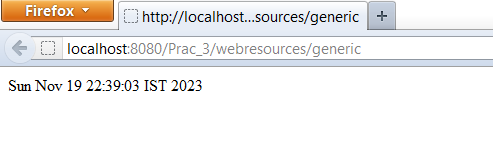


Right Click on Project > Select Clean and Build > Right Click On Project > Select Deploy

Test Your Web Services



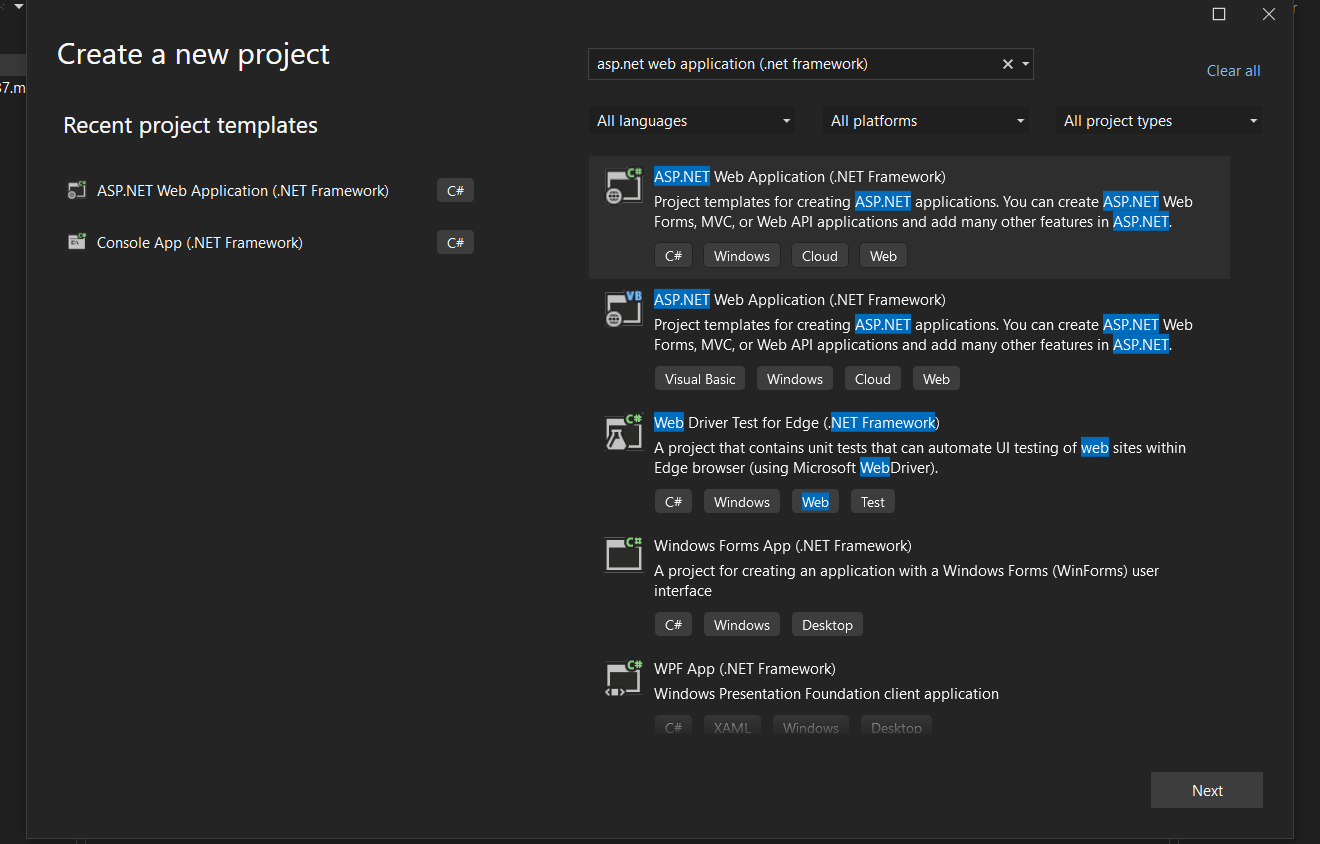
Output :



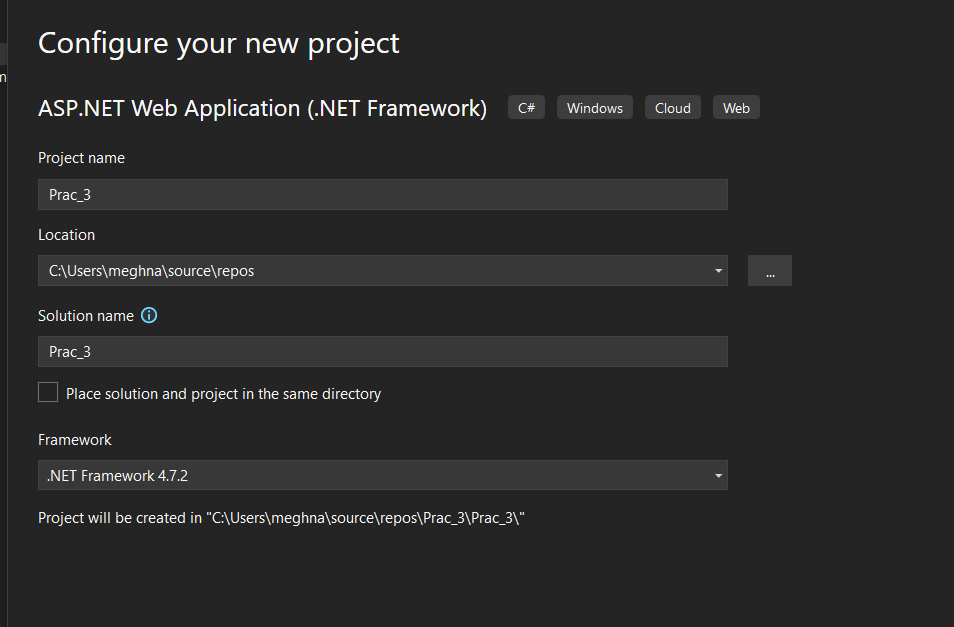
**Prac\_No : 04**

**Title : MVC Application**

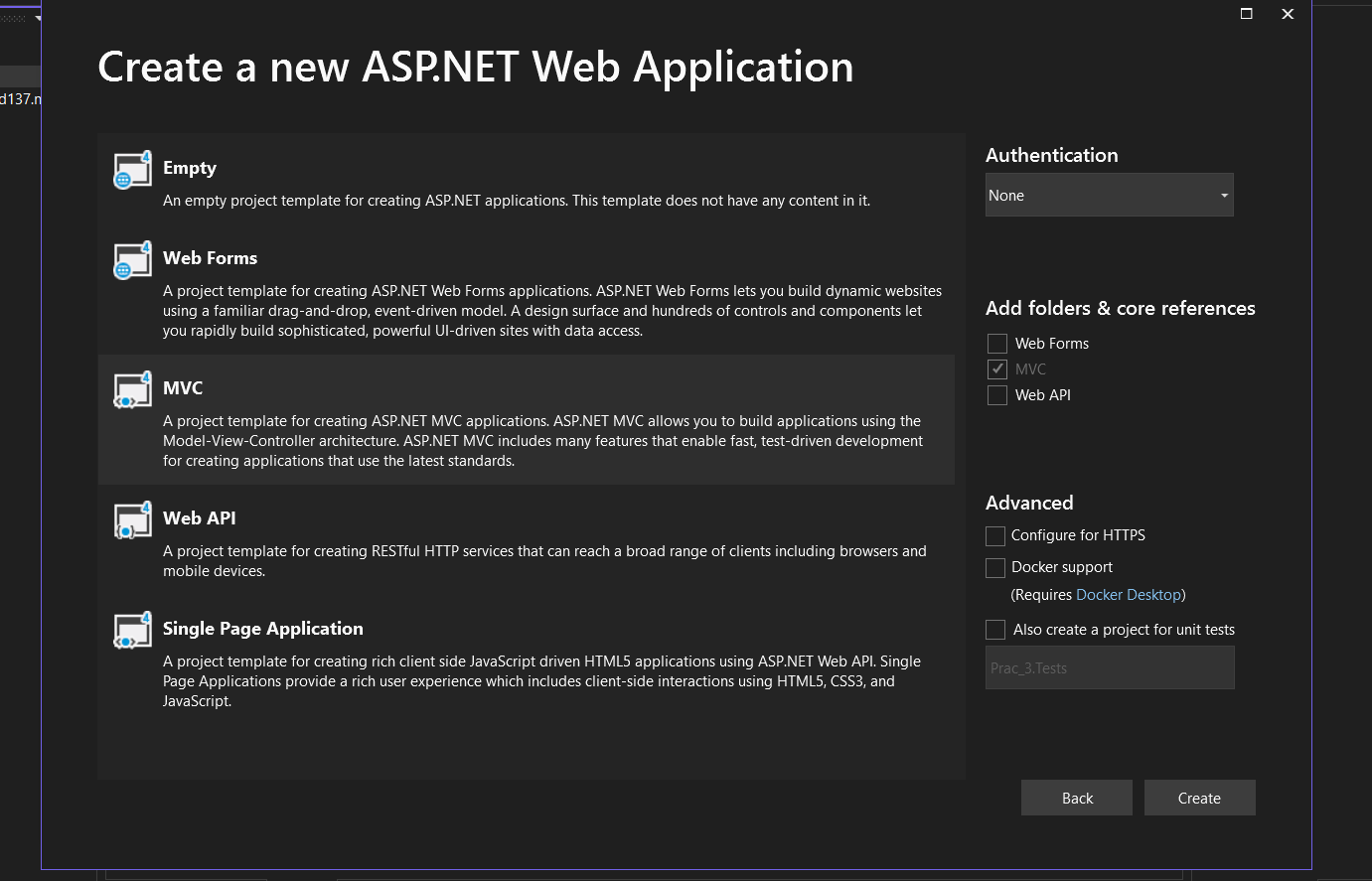
Open Visual Studio > Create New Project > Choose ASP.NET Web Application(.NET Framework)



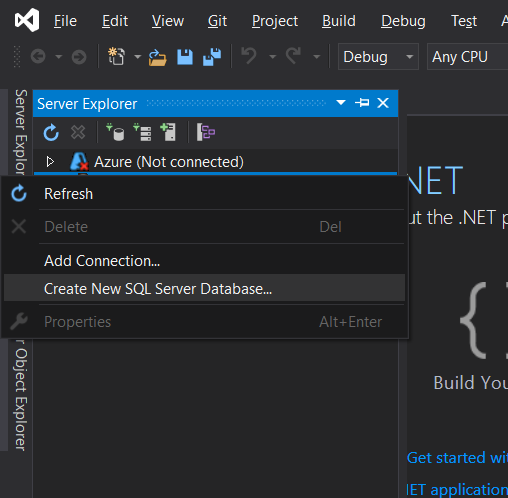
Give Project Name > Create



Choose Web Application (MVC) Template and Unchecked Configure for HTTPS



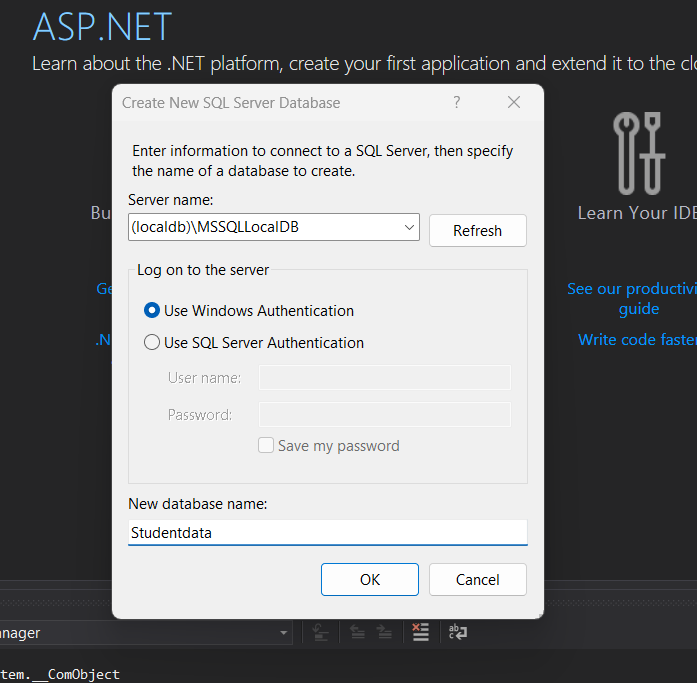
Open View Menu > Select Server Explorer > Right click on Data Connections and add connection



Choose Microsoft SQL Server as Data Source > Continue

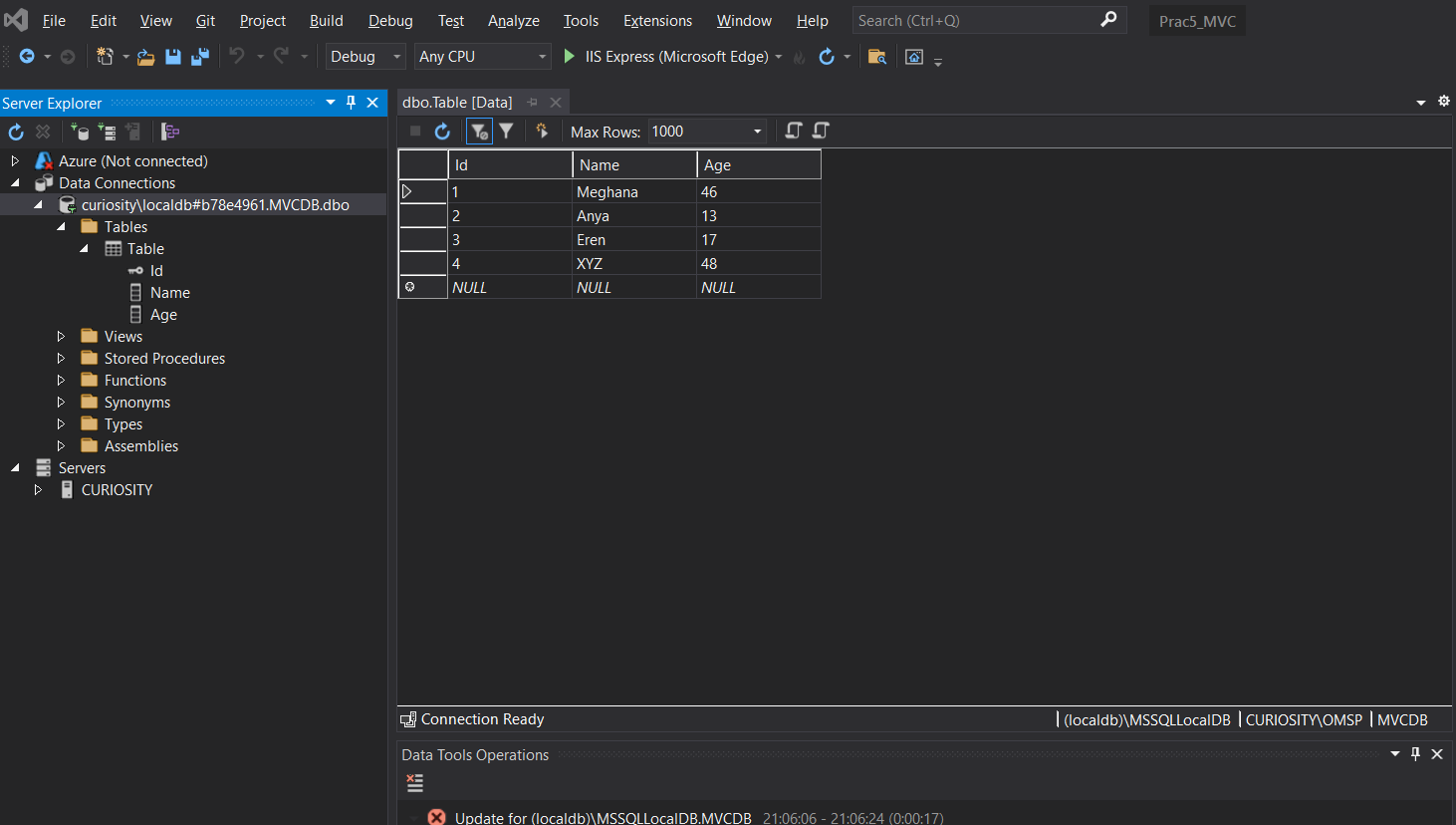
In Data Connection > Choose Create New SQL Database Server

Give Server Name and New Database

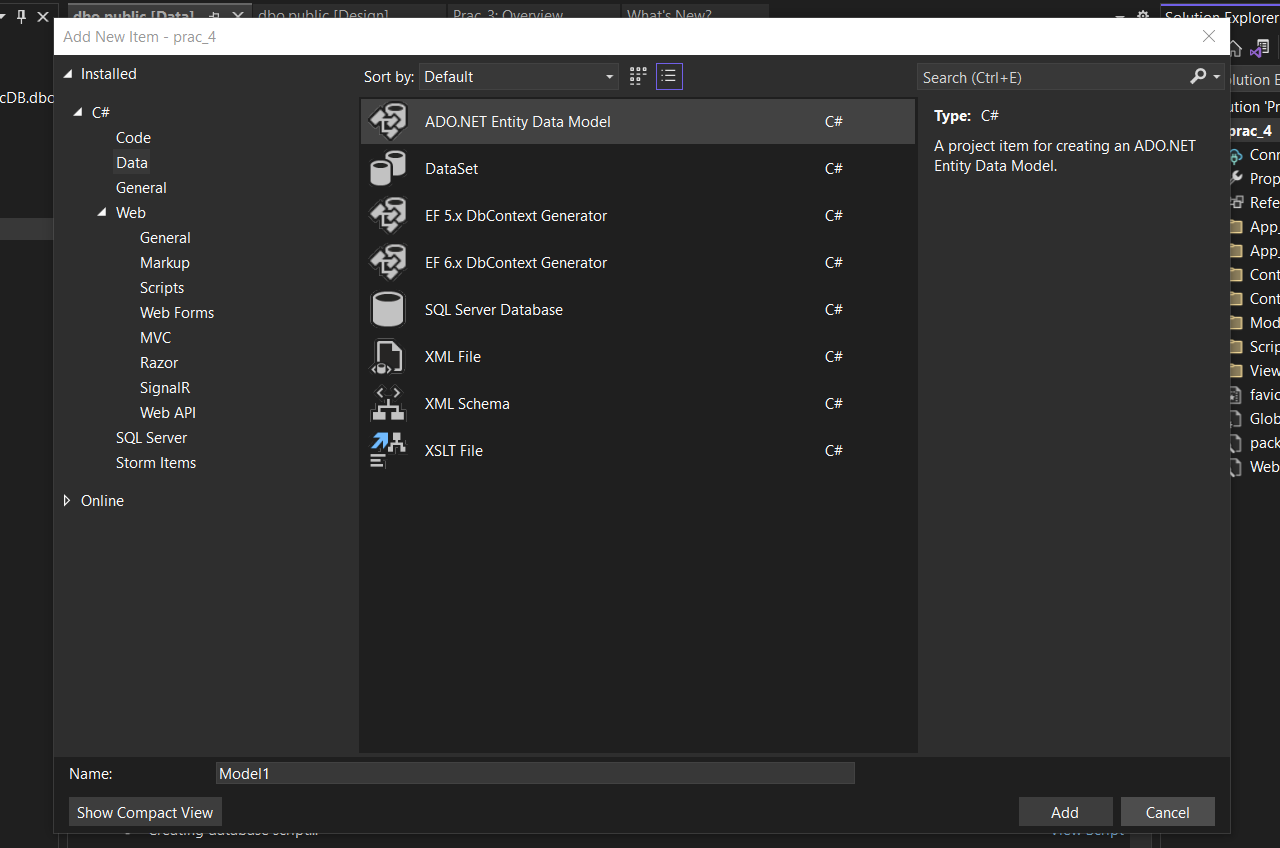


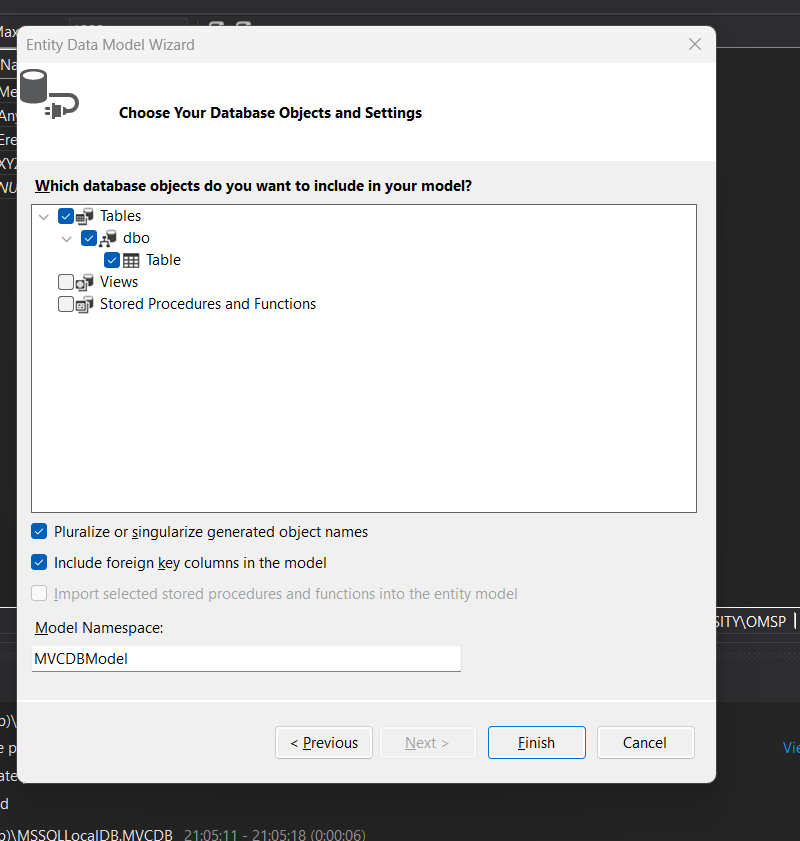
Create Table > Add columns and change table name > Update > Add Data



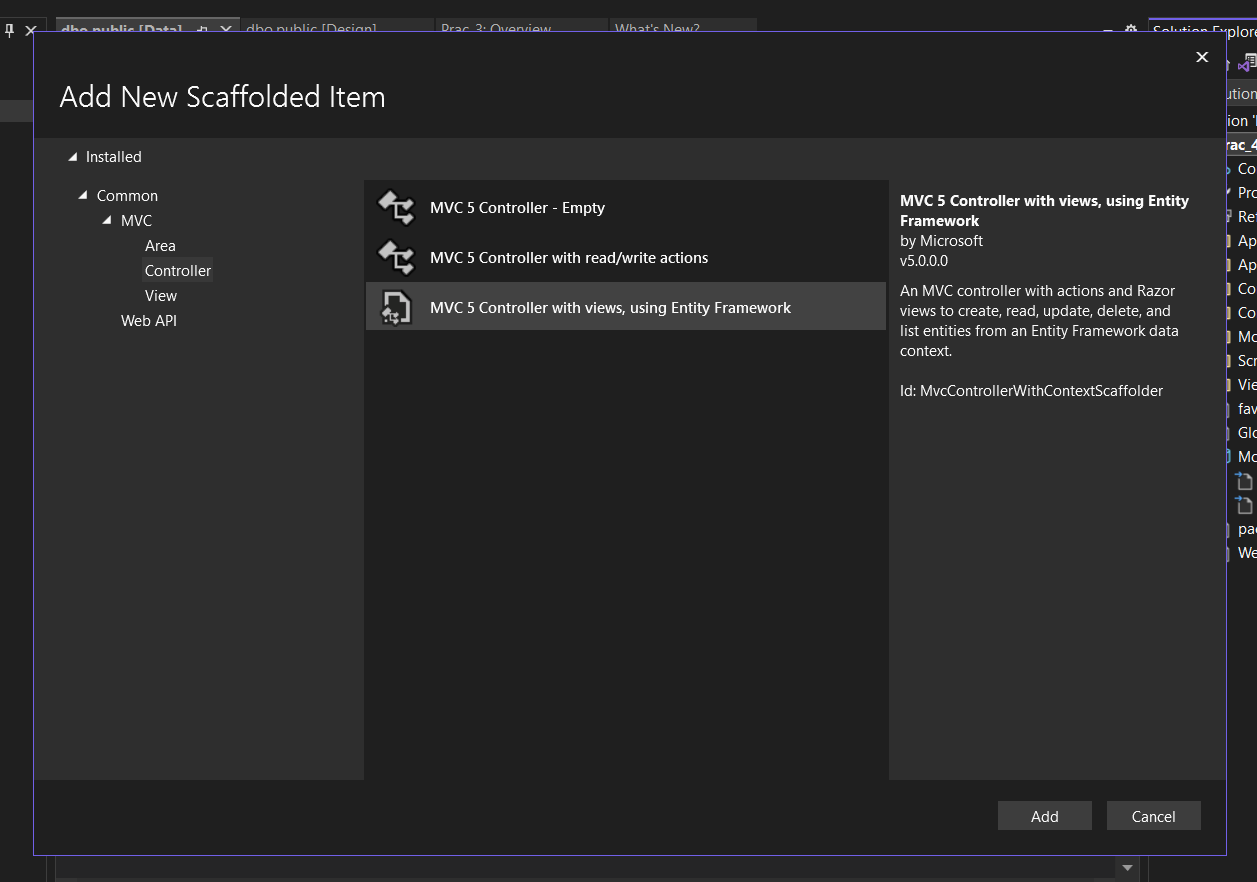


Right Click on Project in Solution Explorer > Add > New Items > Select Data from left panel and Choose ADO.NET Entity Data Model > Add > Select EF designer from database > Select the database you created > Finish

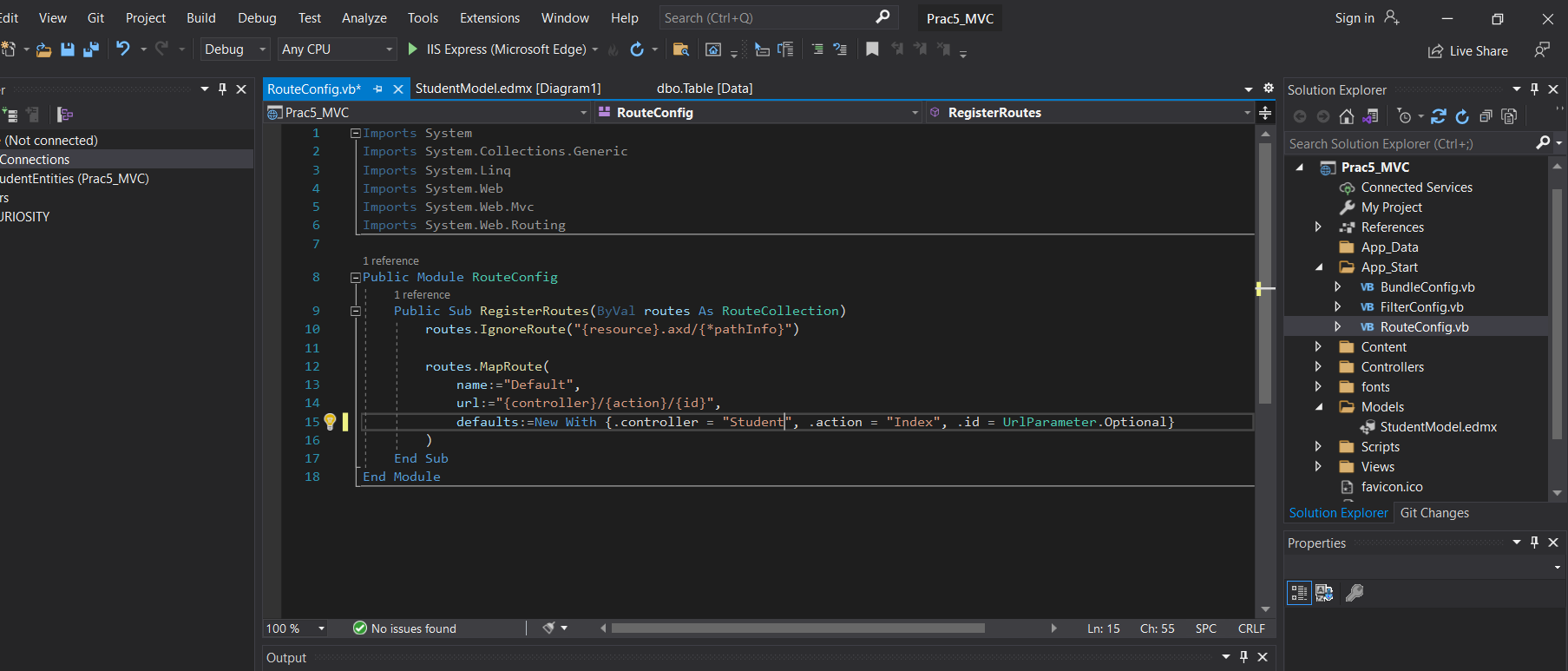




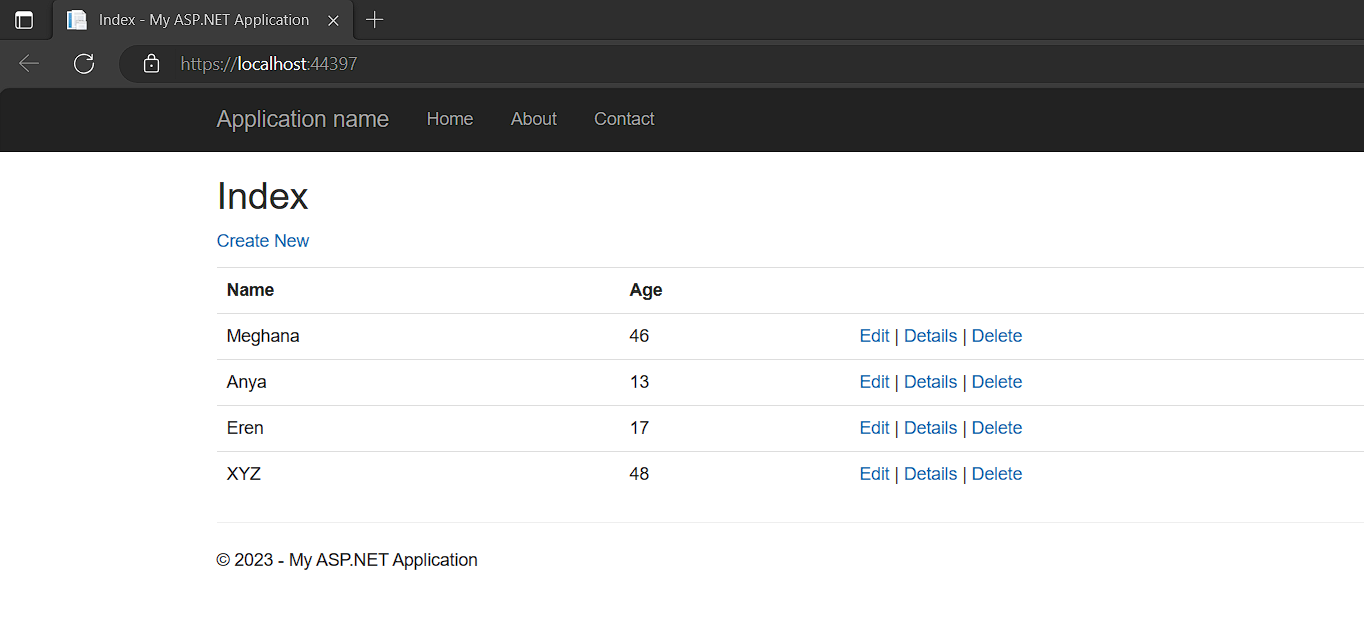
Right Click on project in solution explorer > Add > New Scaffolded Item > Select Controller in left panel and click MVC 5 Controller with read/write action

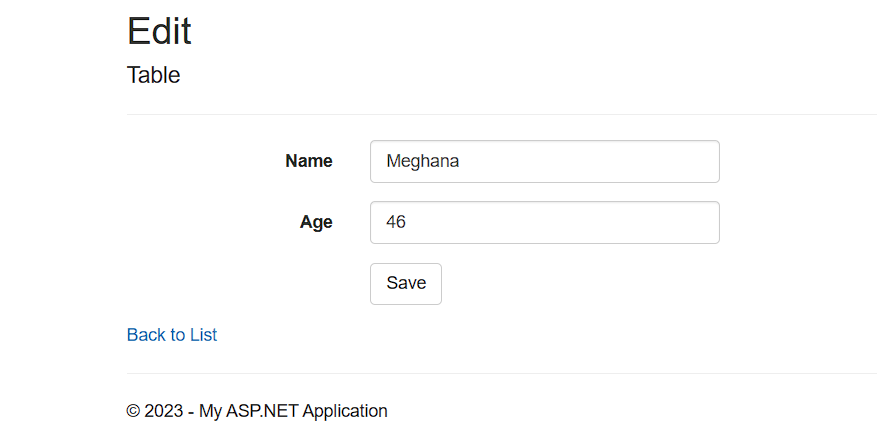


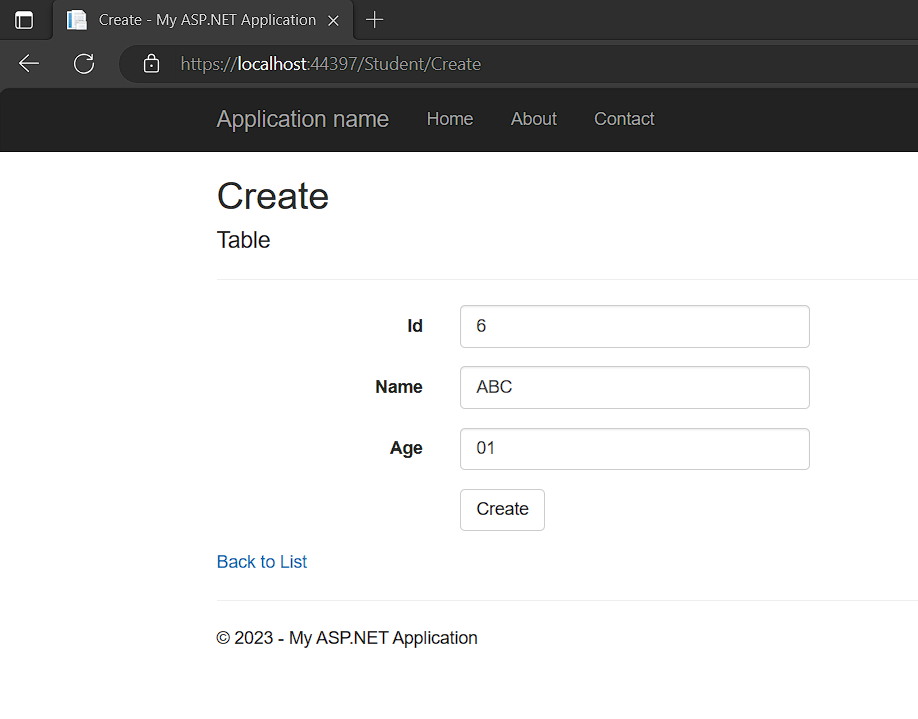
Change Controller name in code

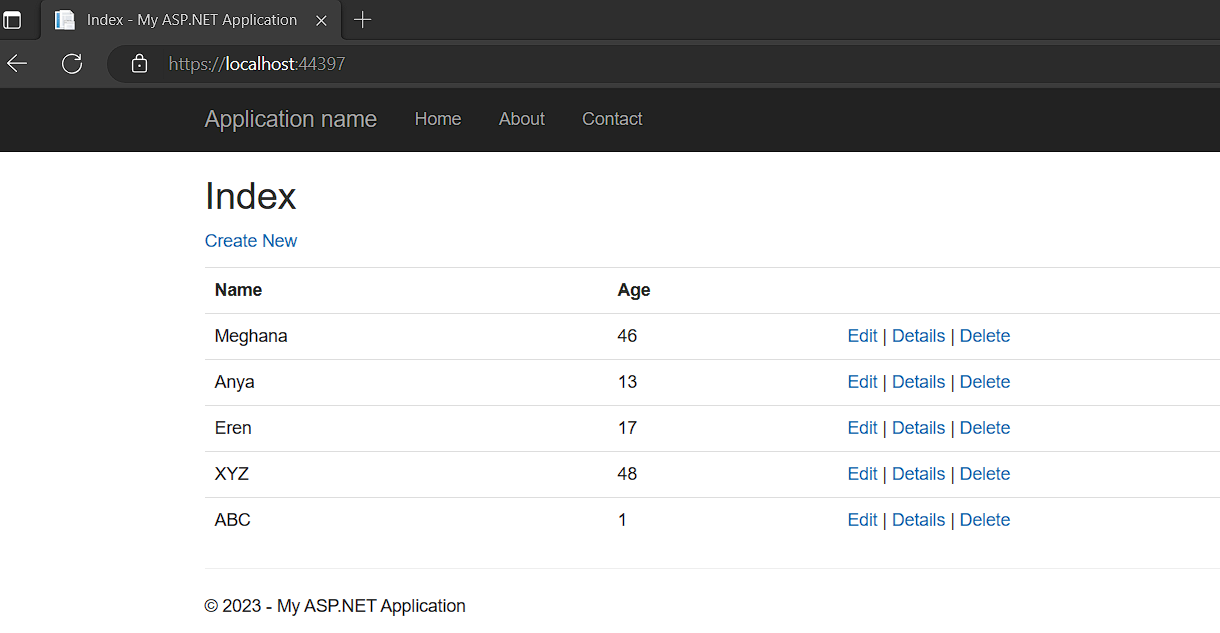


Output:







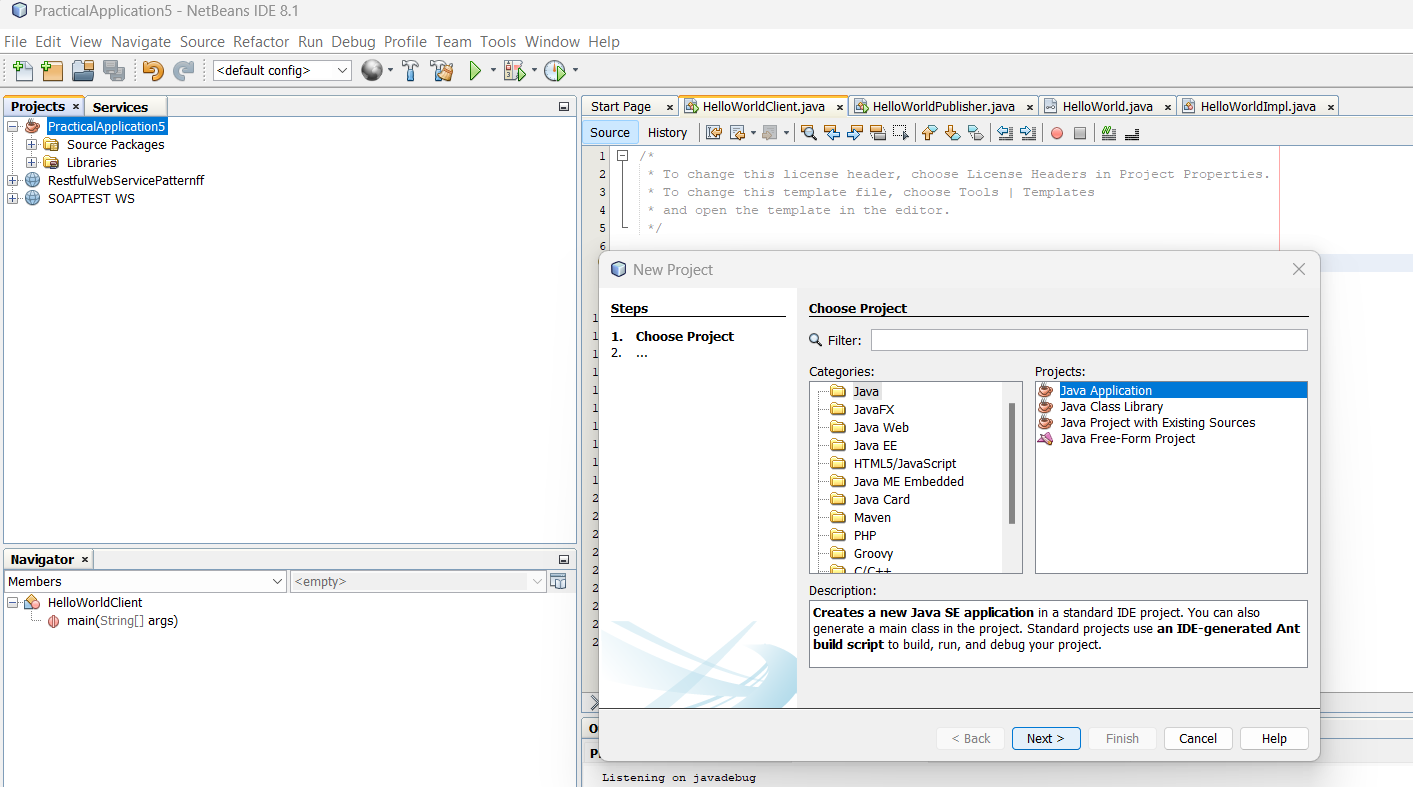


**Prac\_No : 05**

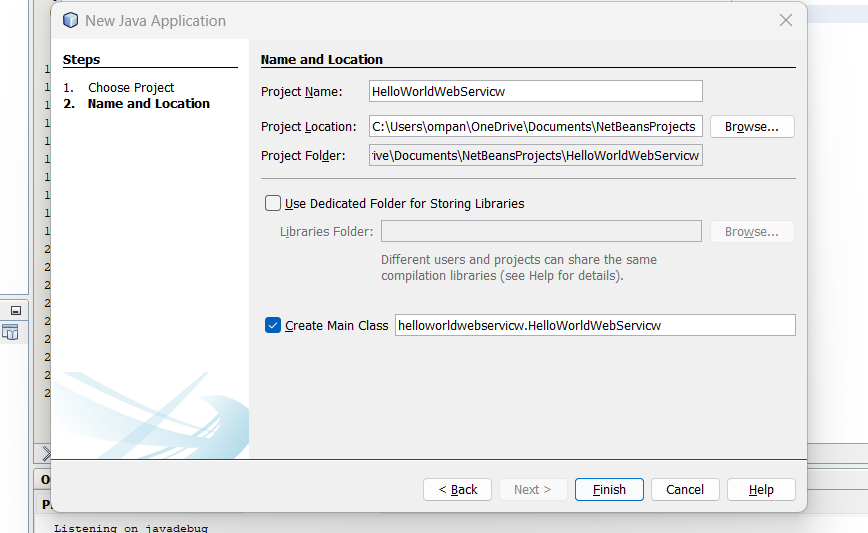
**Title : Service Endpoint**

Step 1: Open NetBeans and Create a New Project

1. Go to File > New Project...
2. In the "New Project" dialog, select "Java" from the categories and "Java Application" as the project type.
3. Click "Next."

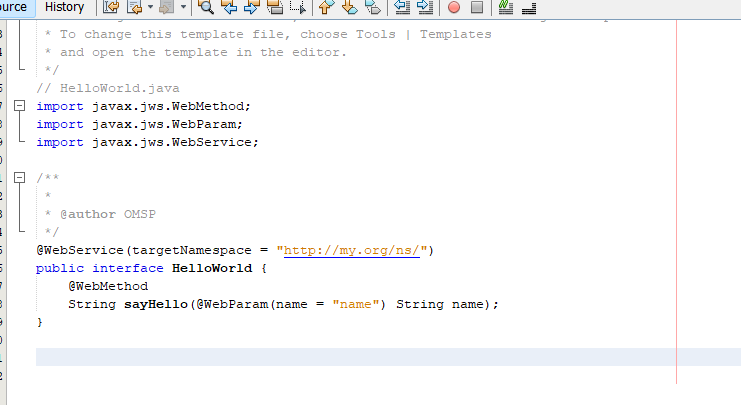


1. Enter a project name, for example, HelloWorldWebService, and specify the project location.
2. Click "Finish."



Step 2: Create HelloWorld Interface

1. Right-click on your project in the "Projects" window.
2. Select New > Java Interface.
3. Enter the interface name as HelloWorld and click "Finish



// HelloWorld.java

import javax.jws.WebMethod;

import javax.jws.WebParam;

import javax.jws.WebService;

@WebService

public interface HelloWorld {

@WebMethod

String sayHello(@WebParam(name = "name") String name);

}

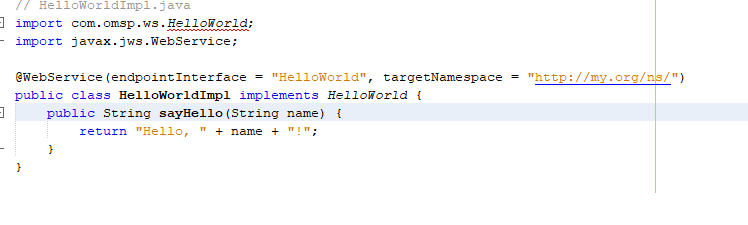
Step 3: Create HelloWorldImpl Class

Right-click on your project.

Select New > Java Class.

Enter the class name as HelloWorldImpl and click "Finish."

Replace the content of the HelloWorldImpl.java file with the content from the previous response.



// HelloWorldImpl.java

import javax.jws.WebService;

@WebService(endpointInterface = "HelloWorld")

public class HelloWorldImpl implements HelloWorld {

@Override

public String sayHello(String name) {

return "Hello, " + name + "!";

}

}

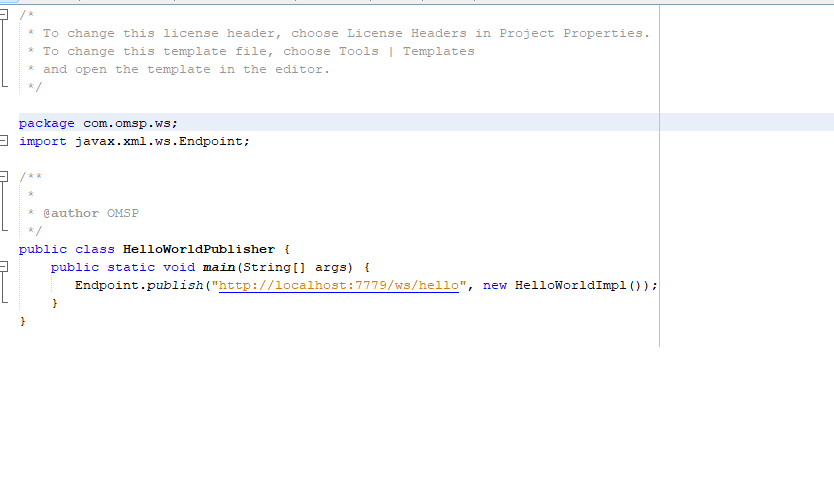
Step 4: Create HelloWorldPublisher Class

Right-click on your project.

Select New > Java Class.

Enter the class name as HelloWorldPublisher and click "Finish."

Replace the content of the HelloWorldPublisher.java file with the content from the previous response.



// HelloWorldPublisher.java

import javax.xml.ws.Endpoint;

public class HelloWorldPublisher {

public static void main(String[] args) {

// Replace "http://localhost:8080/HelloWorld" with your desired endpoint URL

String endpointUrl = "http://localhost:8080/HelloWorld";

Endpoint.publish(endpointUrl, new HelloWorldImpl());

System.out.println("Web service published at: " + endpointUrl);

}

}

Step 5: Compile and Run

Compile all four Java files.

Make sure the HelloWorldPublisher is running.

Run the HelloWorldClient to test the web service.

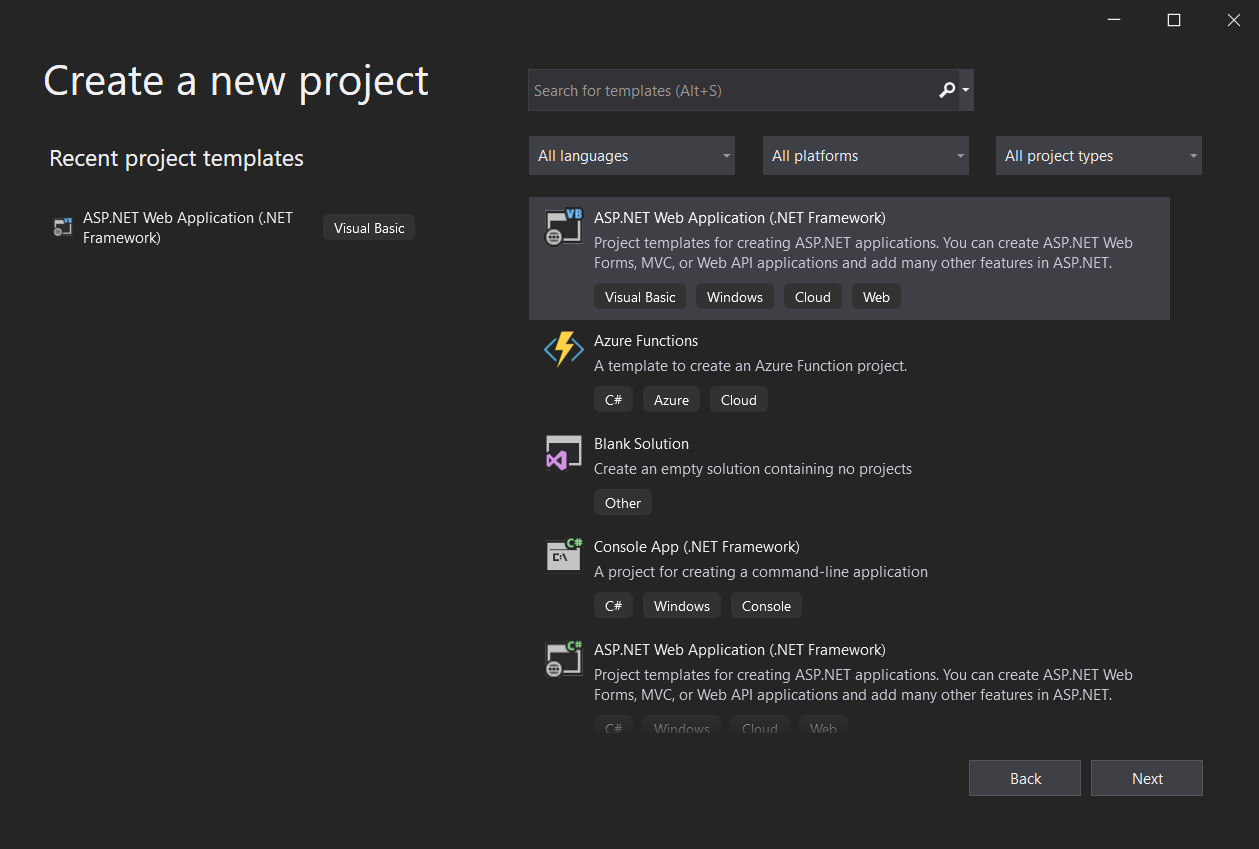
This example uses JAX-WS for creating a SOAP-based web service and client. The HelloWorld interface defines the web service contract, the HelloWorldImpl class implements the actual logic, the HelloWorldPublisher publishes the web service, and the HelloWorldClient is a simple client to test the service

**Prac\_No : 06**

**Title : RestFul Services Using WEB-API**

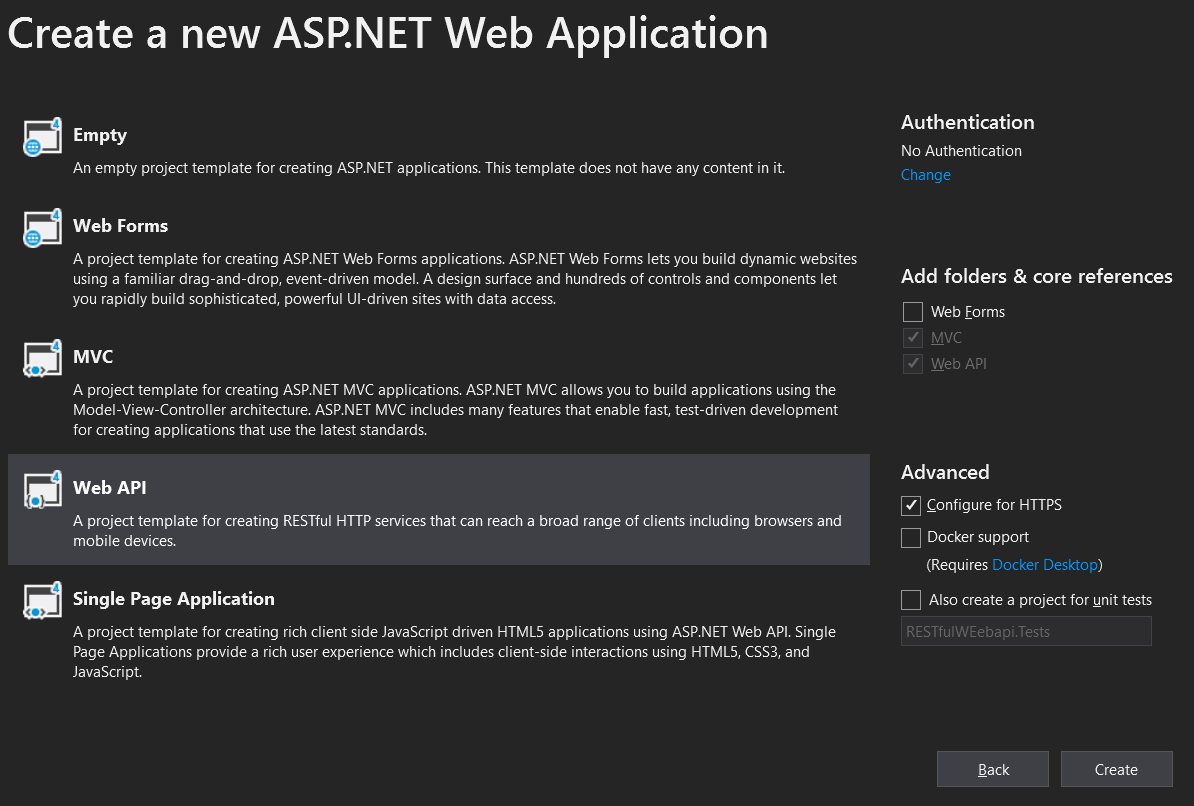
Step 1. Open Visual Studio and Create a New Project:

1. In the "Create a new project" dialog, select "ASP.NET Core Web Application" from the list.
2. Choose a name and location for your project.
3. Select the appropriate framework (e.g., ASP.NET Core 3.1 or later).
4. Choose the "Web Application (Model-View-Controller)" template.
5. Click "Create."



2. Configure the Project:

1. In the "Create a new ASP.NET Core web application" dialog:
2. Select "Configure for HTTPS" if you want to enable secure connections.
3. Choose the authentication type (Individual User Accounts, Windows Authentication, No Authentication).
4. Click "Create."



Step 3. Open Server Explorer:

1.In Visual Studio, go to the "View" menu.

Select "Server Explorer" from the dropdown.

2. Connect to a Database:

In the Server Explorer, right-click on "Data Connections."

Select "Add Connection."

3. Choose Data Source:

In the "Add Connection" dialog, choose the appropriate data source. For a SQL Server database, select "Microsoft SQL Server."

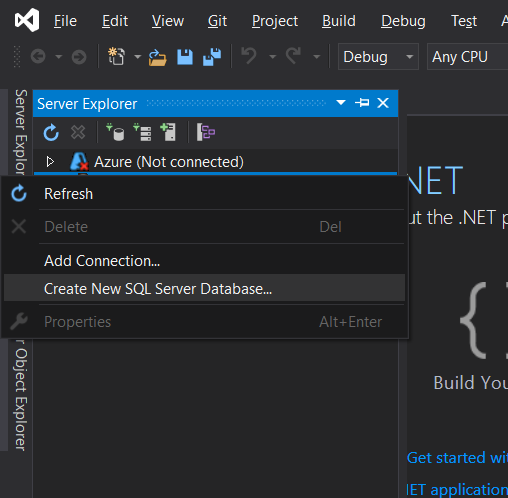
Enter the Server name and choose the authentication method (Windows Authentication or SQL Server Authentication).

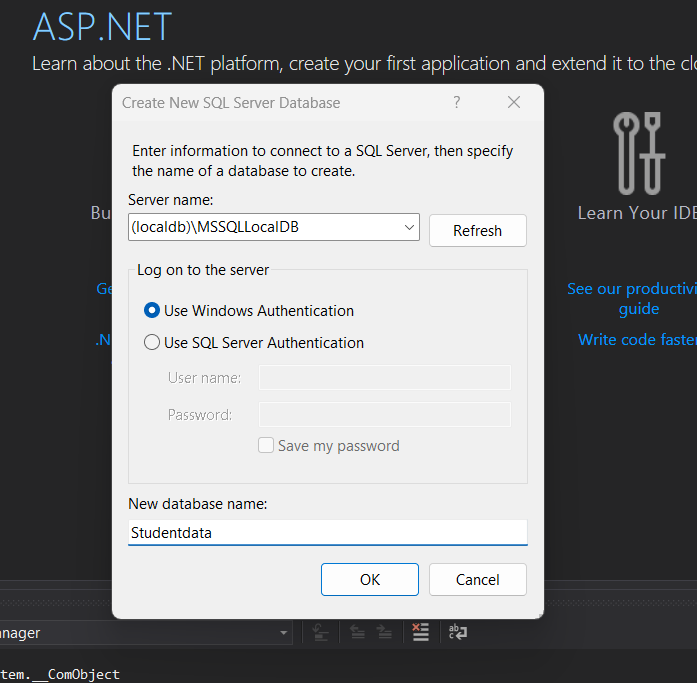
Click "Connect."

4. Select or Create a Database:

Once connected, you will see a list of databases on the server.

You can either select an existing database or create a new one.





Step 5: Creating and designing the table

Step 1: Create a New Table

In the "SQL Server Object Explorer," expand your connected database node.

Right-click on "Tables" and choose "Add New Table."

Step 4: Design the Table

In the "Table Designer," you'll see columns such as "Column Name," "Data Type," etc.

Add a new column:

Click on the first empty row in "Column Name" and enter "ID."

Choose "int" as the data type.

Check the "PK" (Primary Key) box.

Add another column:

Click on the next empty row in "Column Name" and enter "Name."

Choose "nvarchar" as the data type.

Set the "Length" to an appropriate value.

Step 5: Input Data

Input the data into the table directly:

For ID, you can either manually input the values or allow the system to auto-increment.

For Name, input the values "OM," "X," "Y," "Z," and leave NULL for the last row.

Step 2: Save the Table

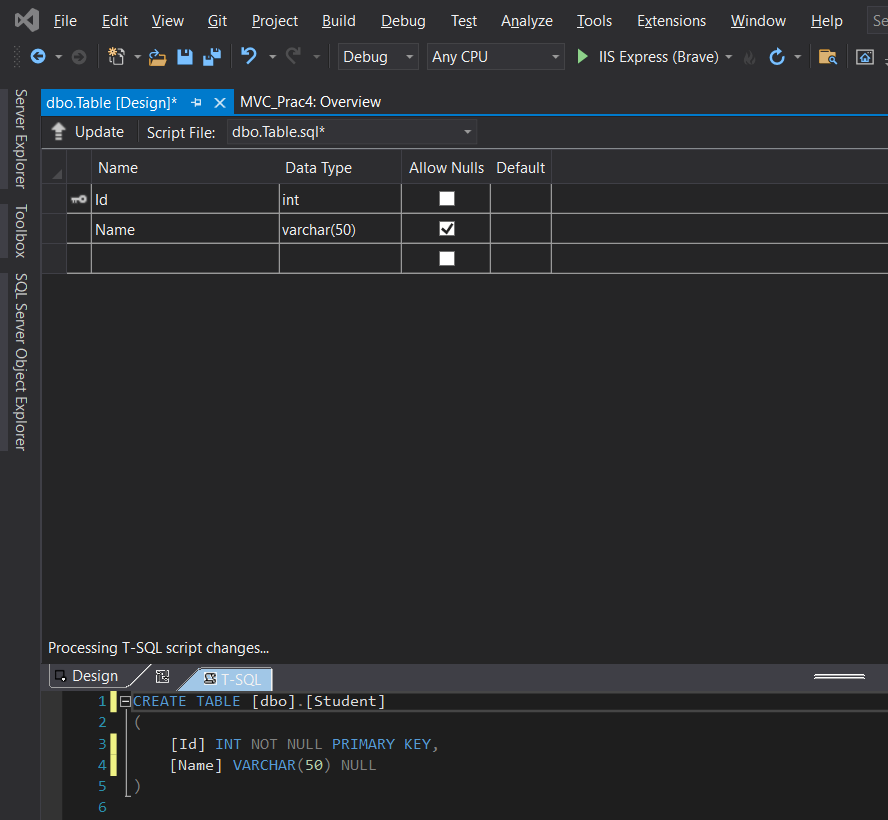
Click the "Update" button in the "Table Designer" toolbar (or press Ctrl + S).

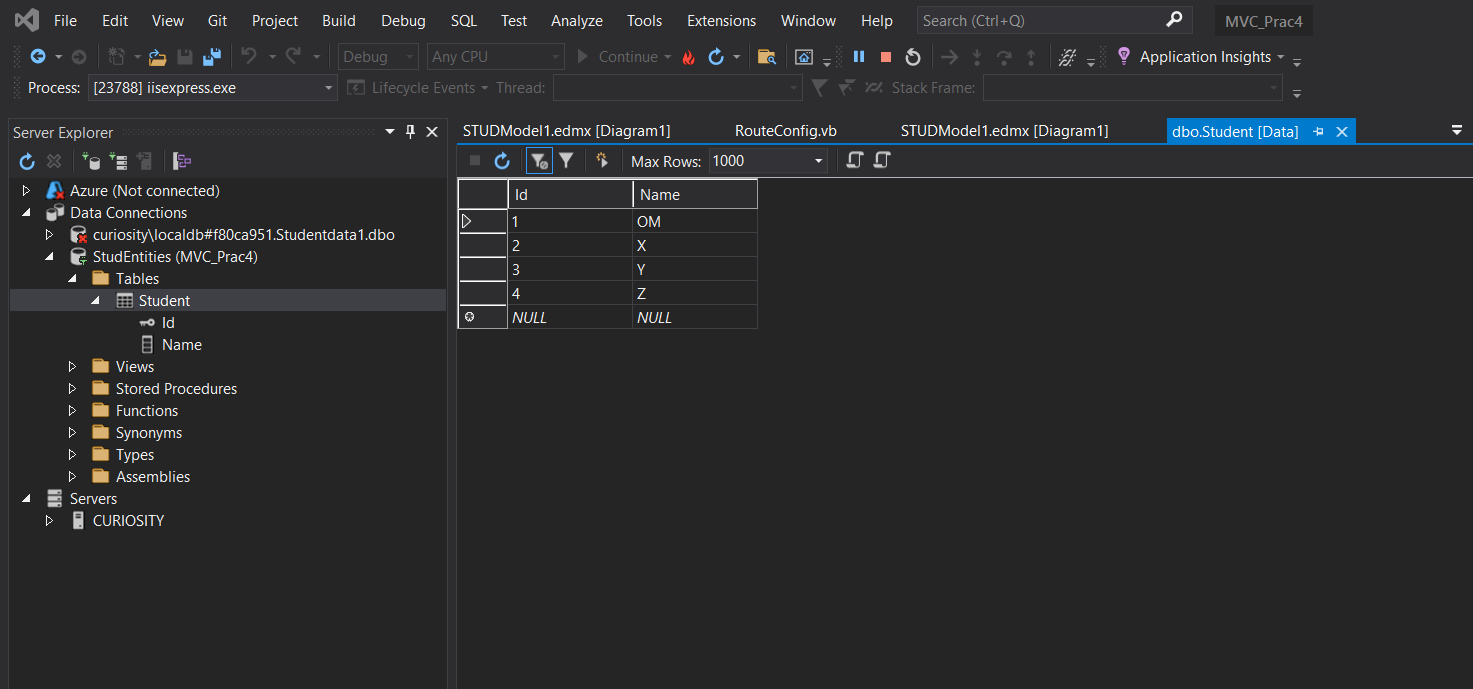
Provide a name for the table and click "OK."

Step 3: Check the Table

In the "SQL Server Object Explorer," refresh the "Tables" node.

You should see your newly created table with data.





Step 6: Adding New ADO.NET Entity Data Model

Step 1: Create a New ADO.NET Entity Data Model

Right-click on your project in Solution Explorer.

Select Add > New Item...

Choose "Data" from the left pane, and then choose "ADO.NET Entity Data Model."

Enter a name for your model (e.g., SchoolModel) and click "Add."

Step 2: Choose Data Connection

Choose "EF Designer from Database" and click "Next."

Select the data connection you previously established and click "Next."

Step 3: Choose Database Objects

Choose the tables or views you want to include in the model (e.g., the table you created with "ID" and "Name").

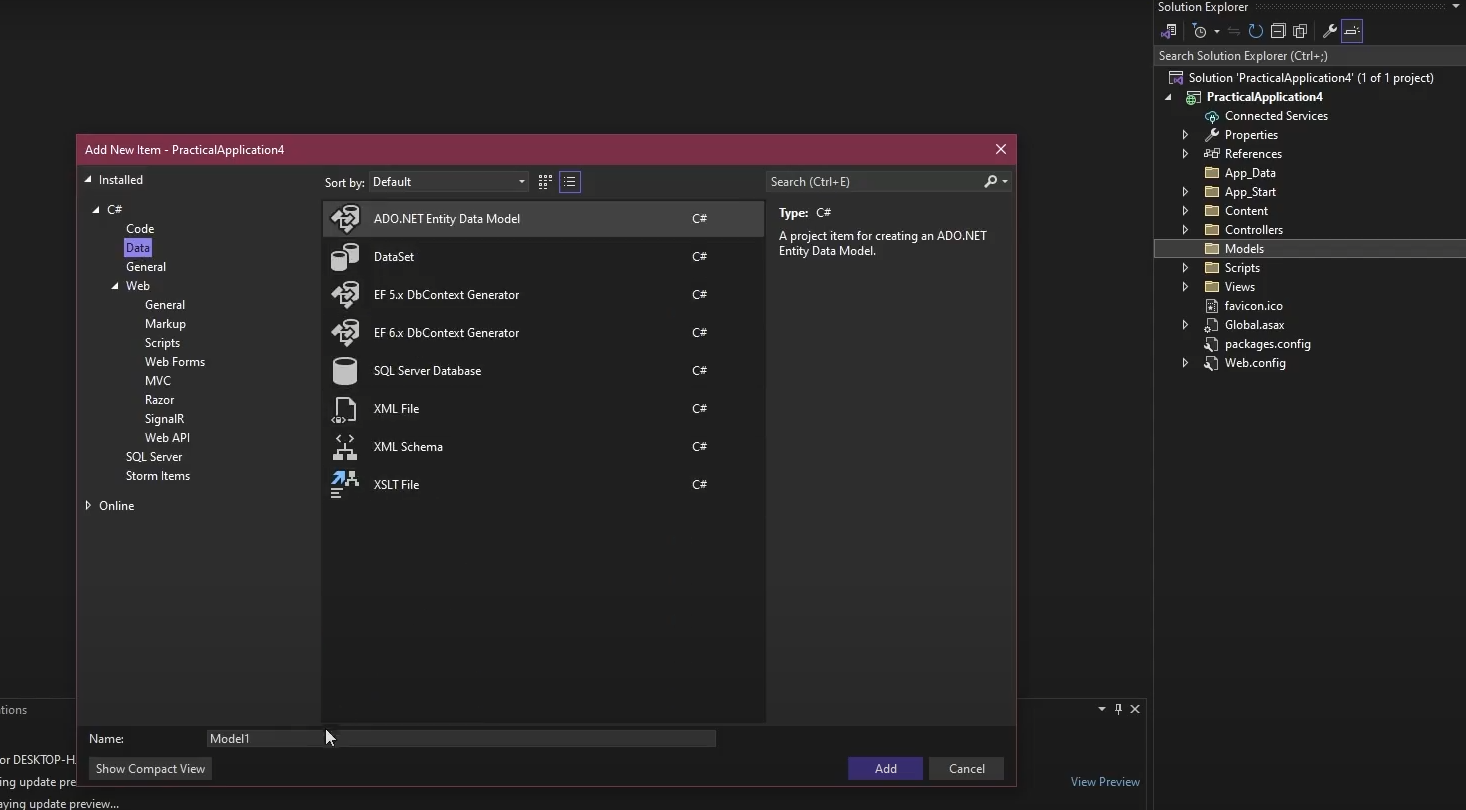
Click "Finish."

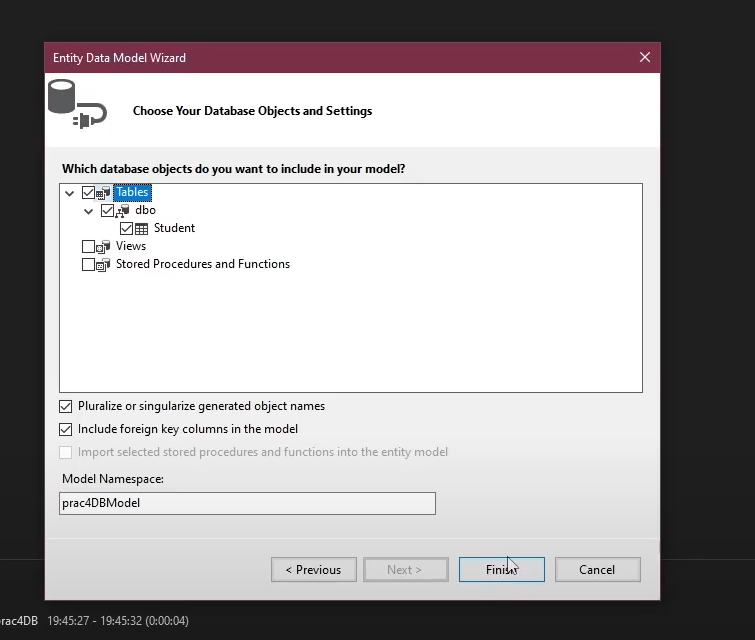
Step 4: Save and Generate Model

Save the model by clicking "Ctrl + S" or selecting File > Save All.

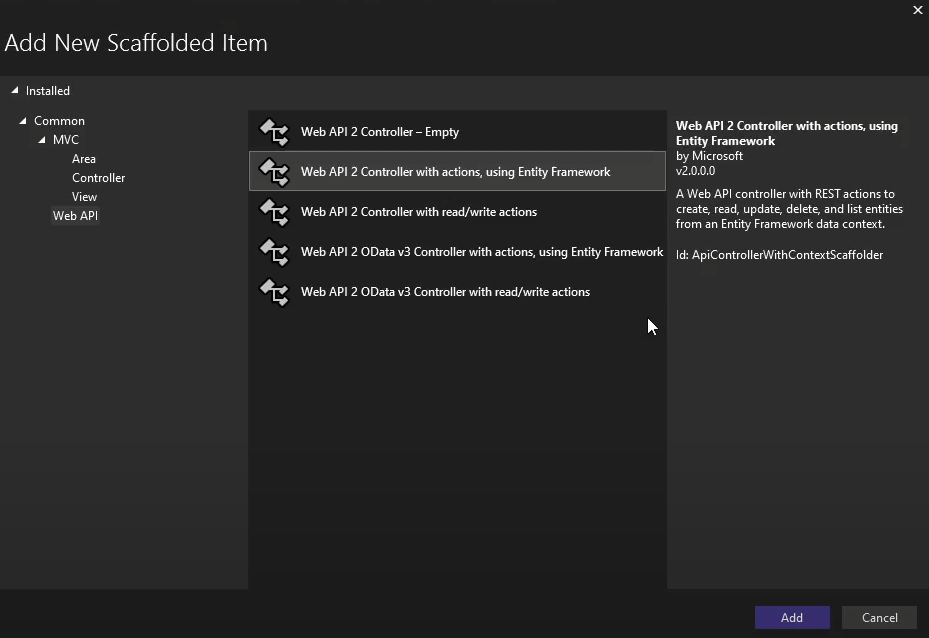
The Entity Data Model will be generated based on the selected database objects.

Now, you have an ADO.NET Entity Data Model in your project. You can use this model to perform CRUD operations on your database through Entity Framework.





Adding Scaffolding item,



﻿using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using System.Web.Http.Description;

using PracticalApplication6.Models;

namespace PracticalApplication6.Controllers

{

public class StudentController : ApiController

{

private prac4DBEntities db = new prac4DBEntities();

// GET: api/Student

public IQueryable<Student> GetStudents()

{

return db.Students;

}

// GET: api/Student/5

[ResponseType(typeof(Student))]

public IHttpActionResult GetStudent(int id)

{

Student student = db.Students.Find(id);

if (student == null)

{

return NotFound();

}

return Ok(student);

}

// PUT: api/Student/5

[ResponseType(typeof(void))]

public IHttpActionResult PutStudent(int id, Student student)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != student.Id)

{

return BadRequest();

}

db.Entry(student).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!StudentExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return StatusCode(HttpStatusCode.NoContent);

}

// POST: api/Student

[ResponseType(typeof(Student))]

public IHttpActionResult PostStudent(Student student)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

db.Students.Add(student);

try

{

db.SaveChanges();

}

catch (DbUpdateException)

{

if (StudentExists(student.Id))

{

return Conflict();

}

else

{

throw;

}

}

return CreatedAtRoute("DefaultApi", new { id = student.Id }, student);

}

// DELETE: api/Student/5

[ResponseType(typeof(Student))]

public IHttpActionResult DeleteStudent(int id)

{

Student student = db.Students.Find(id);

if (student == null)

{

return NotFound();

}

db.Students.Remove(student);

db.SaveChanges();

return Ok(student);

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

private bool StudentExists(int id)

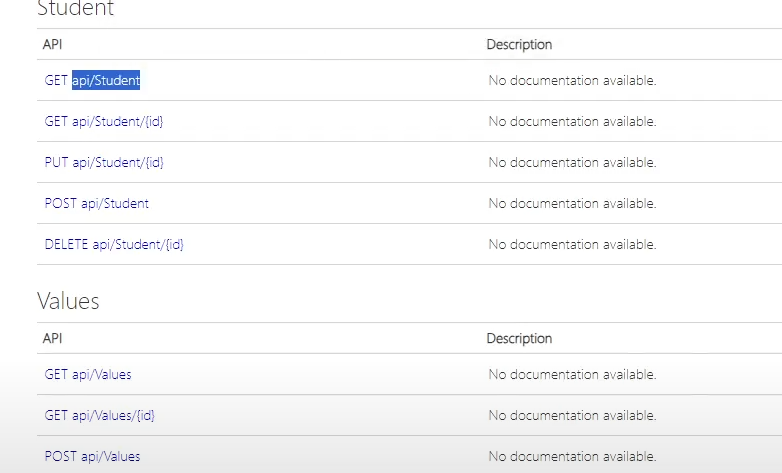
{

return db.Students.Count(e => e.Id == id) > 0;

}

}

}



**Prac\_No : 07**

**Title : Web Application Using Azure**

Create a New Project:

Go to File > New > Project....

In the "Create a new project" dialog, select the "Cloud" category on the left.

Choose "Azure Cloud Service" from the list of templates.

Choose the "ASP.NET Web Role" template.

Set the project and solution name, and choose a location to save the project.

Click "Create."

Configure Cloud Service Project:

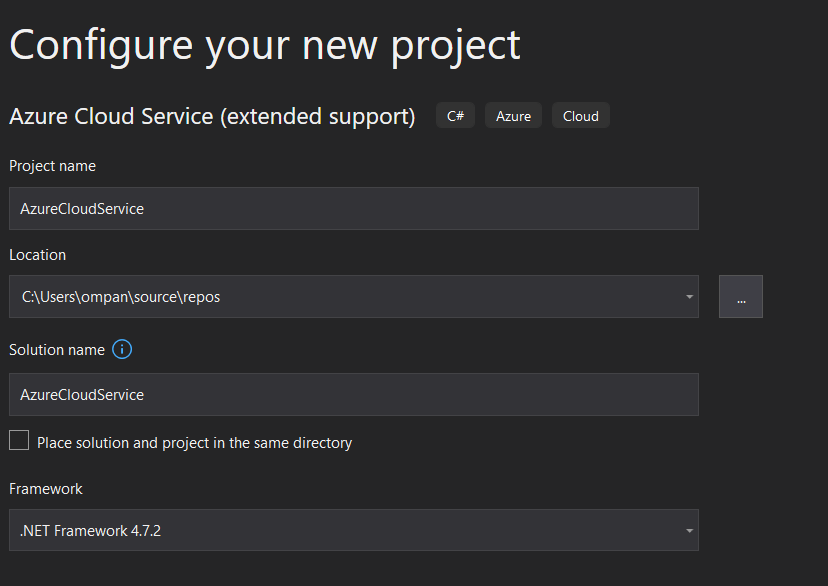
In the Solution Explorer, you'll see two projects: one for the cloud service and one for the web role.

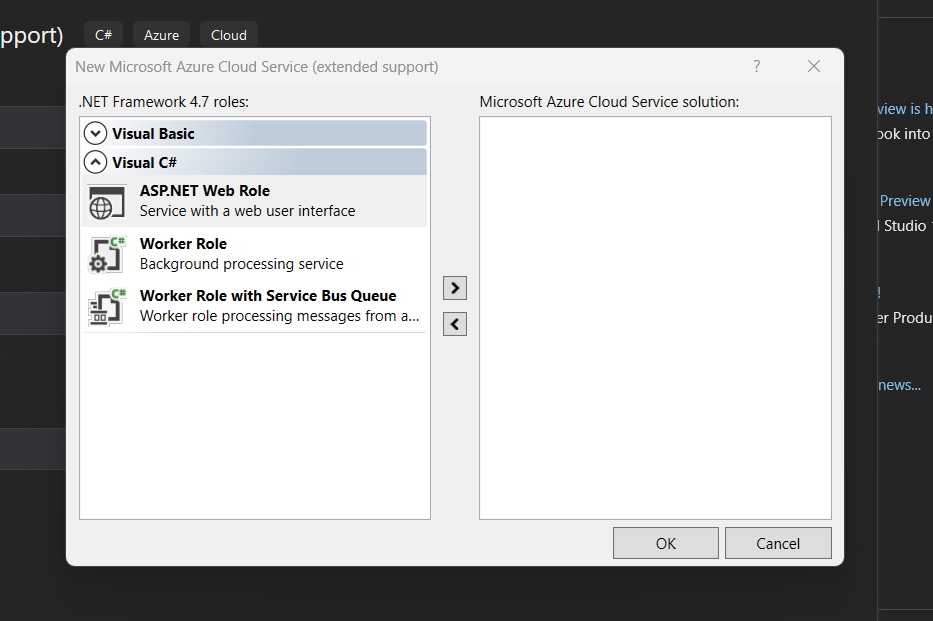
Right-click on the cloud service project and select "Properties."

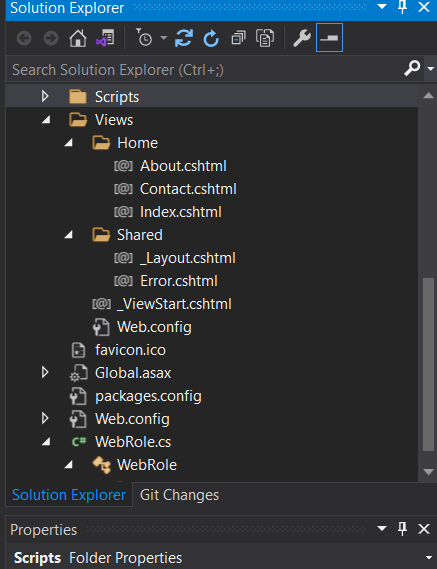
Configure the roles, settings, and other options as needed.

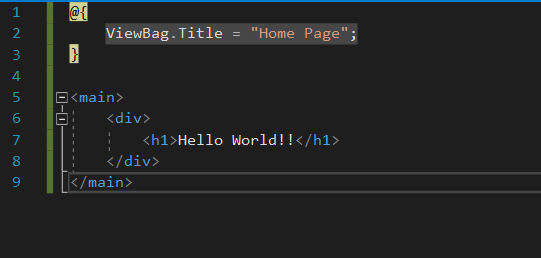
Write "Hello World" Code:

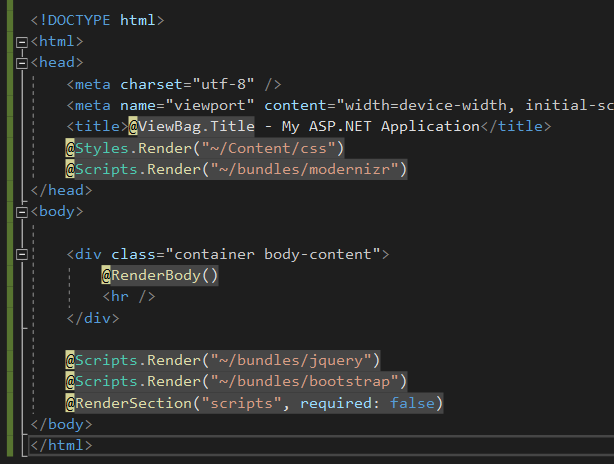
Open the Default.aspx file in the web role project.

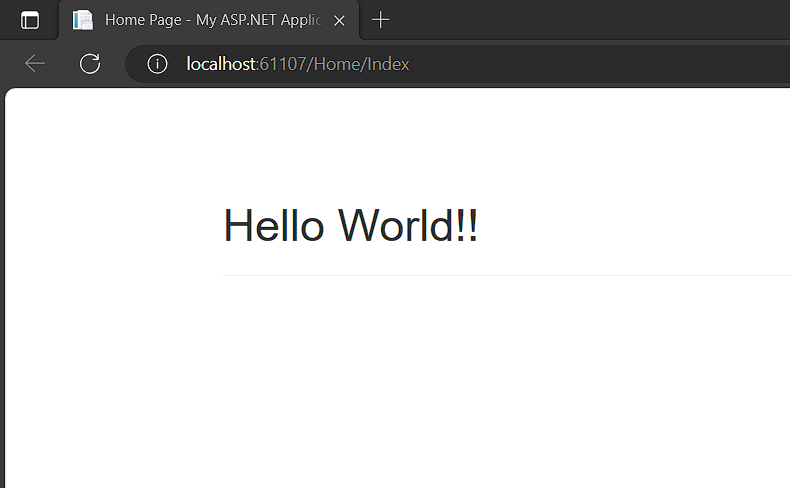
Replace the existing content with a simple "Hello World" message. For example:

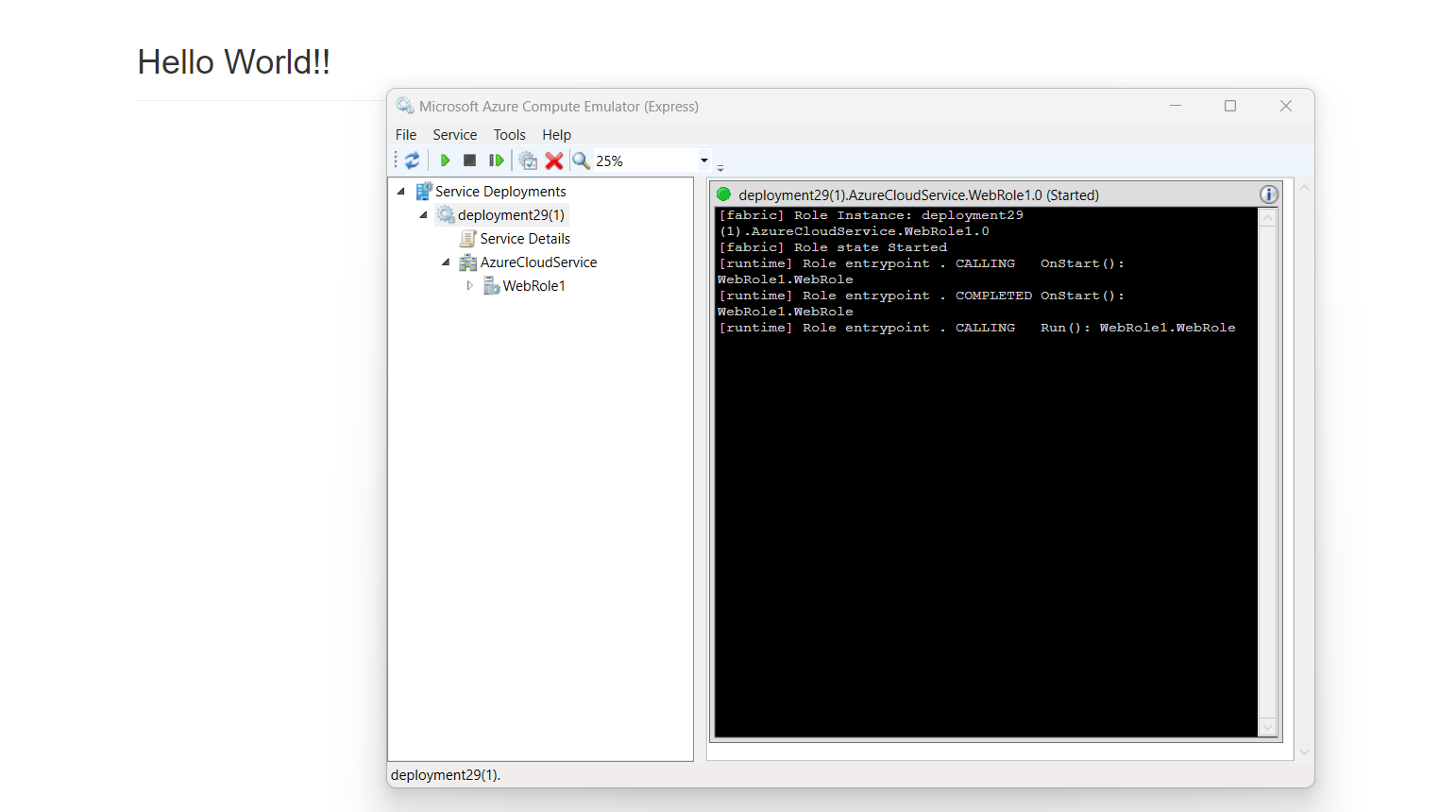










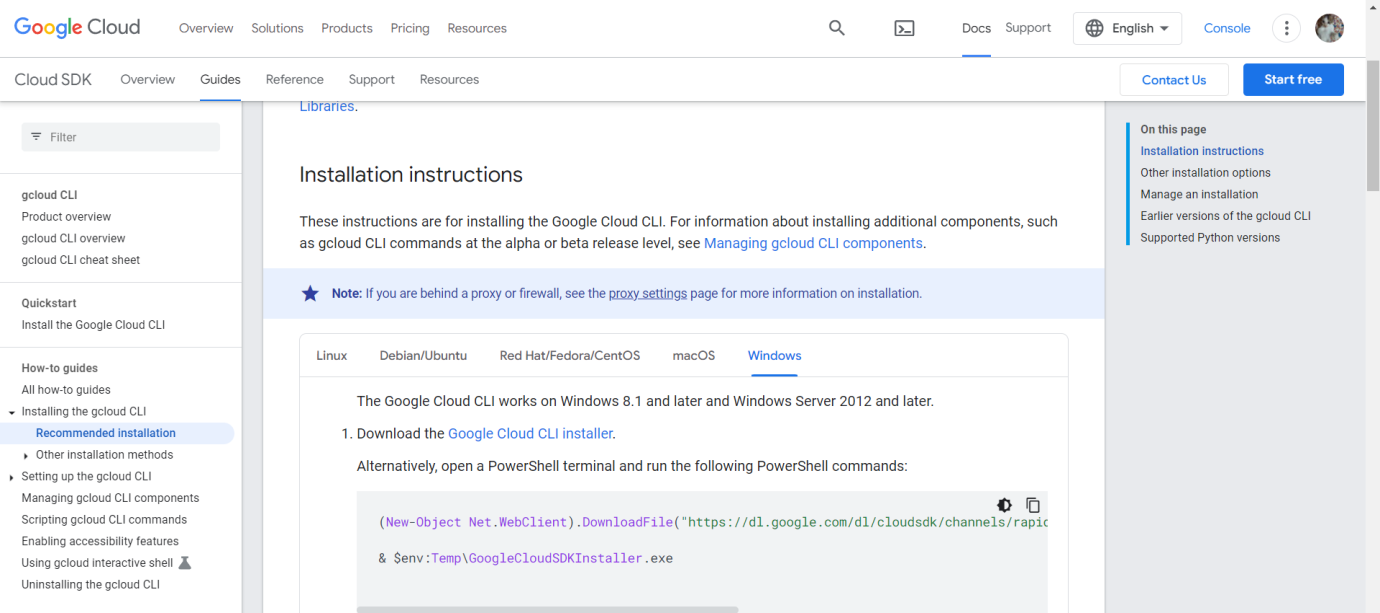


**Prac\_No : 08**

**Title : Install Google App Engine. Create hello world app and other simple web applications using python**

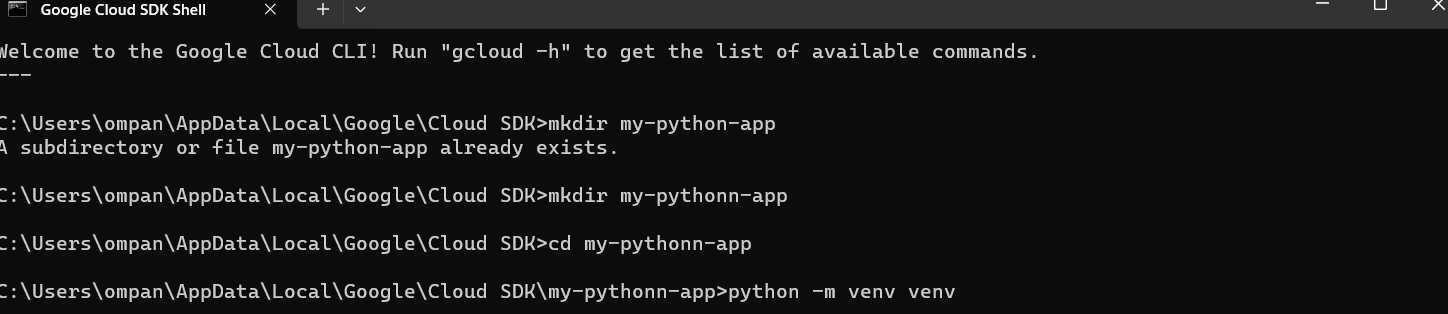
Step 1: Install Google Cloud SDK

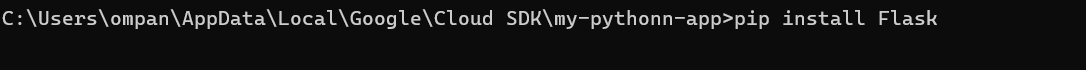
If you haven't installed the Google Cloud SDK, you can download it from [here](https://cloud.google.com/sdk/docs/install).



Step 2: Create a "Hello, World!" Python App

Create a New Directory: Open your terminal or command prompt and create a new directory for your project



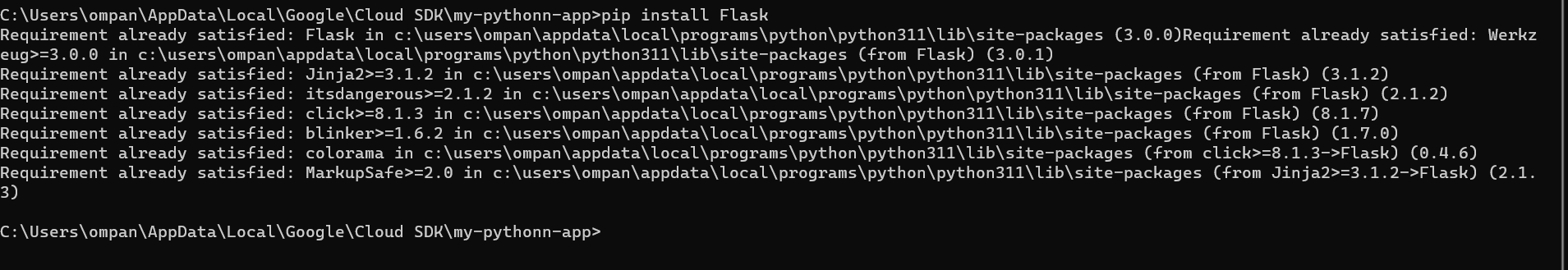


mkdir my-python-app

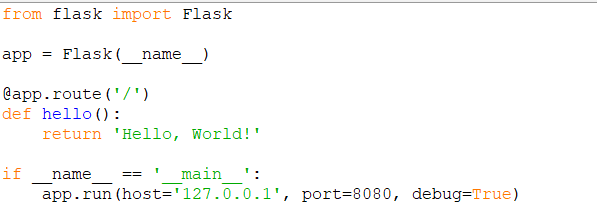
cd my-python-app

Step 3 - Install Flask: Install Flask, a lightweight web application framework for Python.

pip install Flask



Step 4 :Create the Flask App: Create a file named app.py with the following content:



from flask import Flask

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

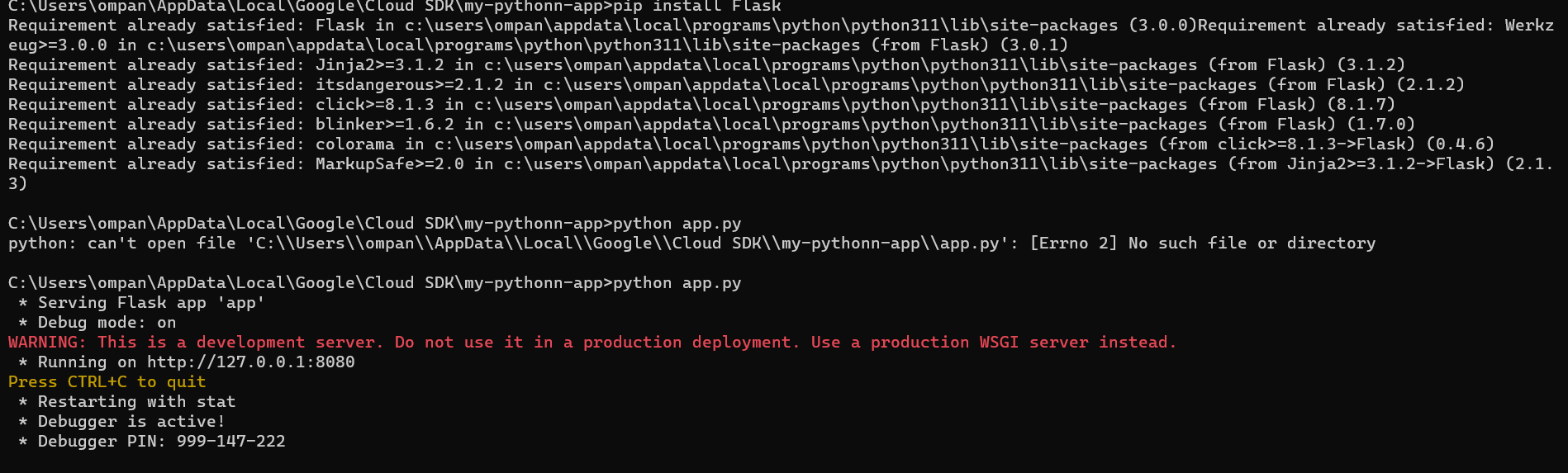
@app.route('/')

def hello():

return 'Hello, World!'

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

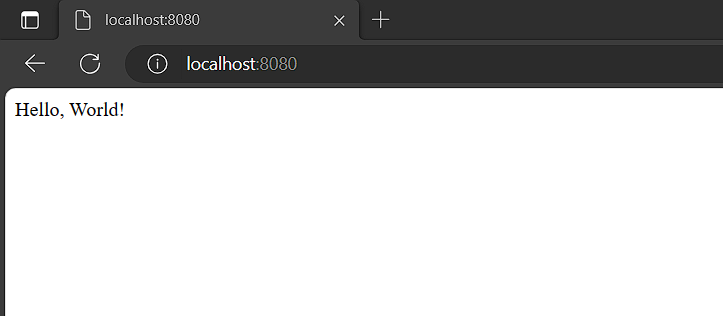
app.run(host='127.0.0.1', port=8080, debug=True)



Step 5: Test the App Locally

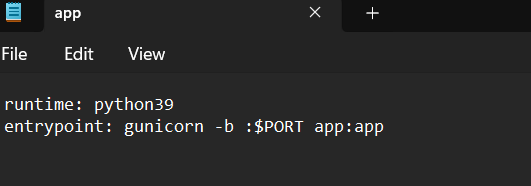
Run the Flask app locally to make sure it works

http://localhost:8080 in your web browser. You should see "Hello, World!"



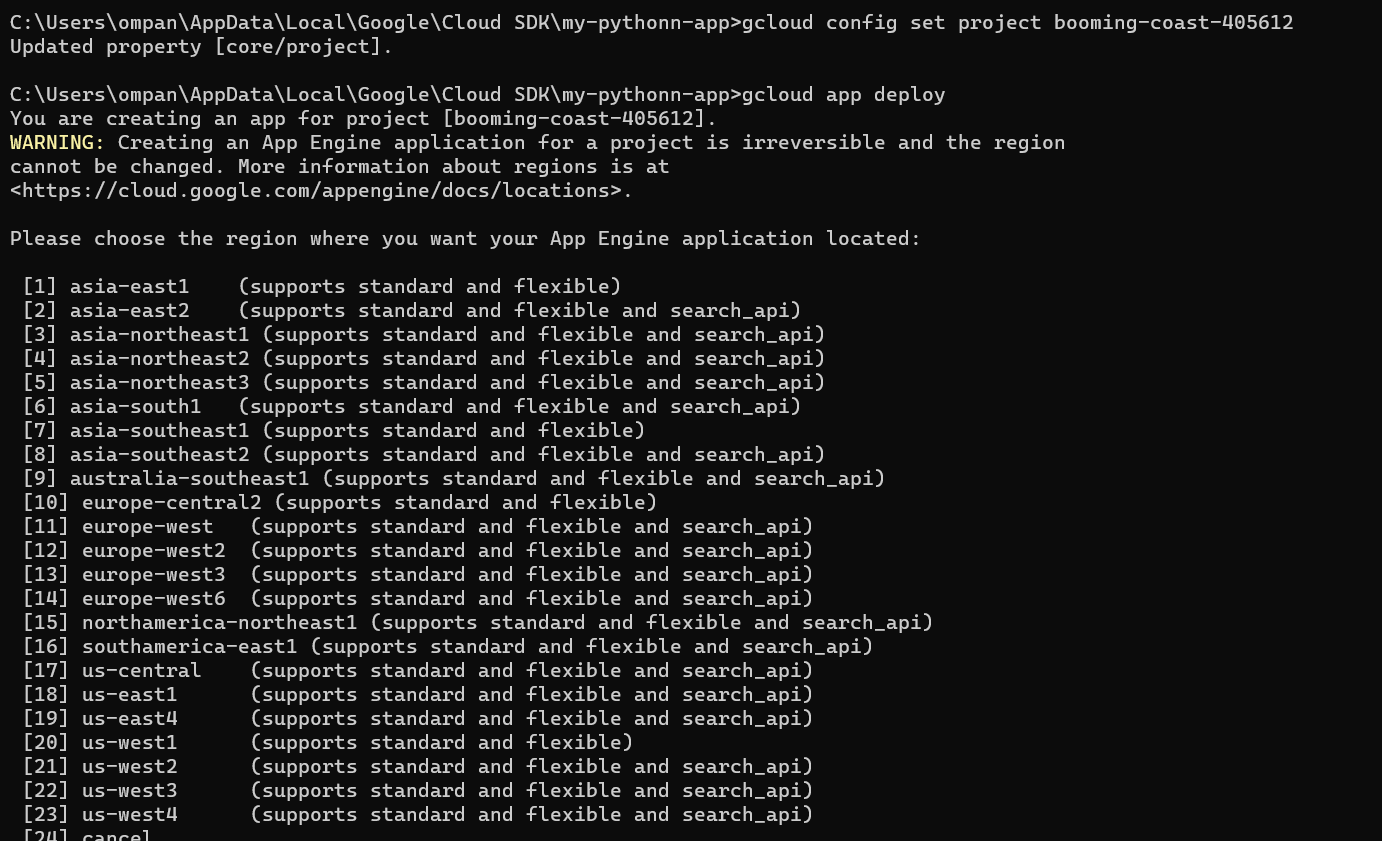
Step 6: Deploy to Google App Engine

Create app.yaml: Create a file named app.yaml in your project directory with the following content:



Deploy to App Engine: Deploy your app to Google App Engine.

Command - gcloud app deploy



STEP 7

Open the [Google Cloud Console](https://console.cloud.google.com/).

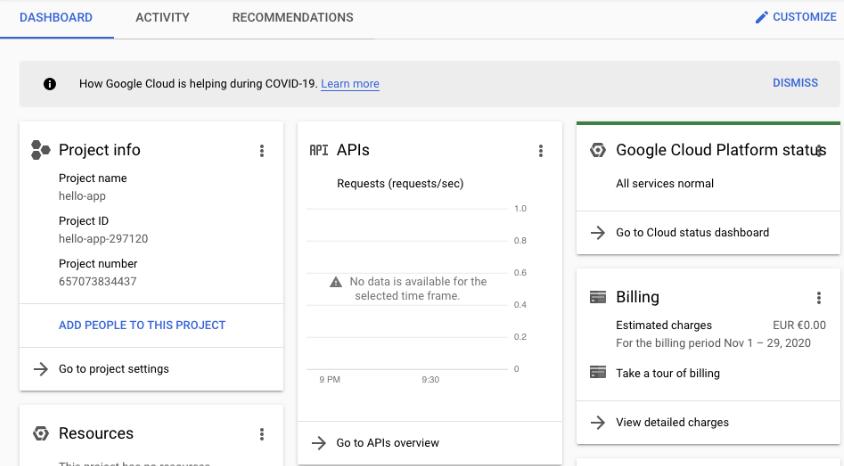
Navigate to "IAM & Admin" > "IAM."

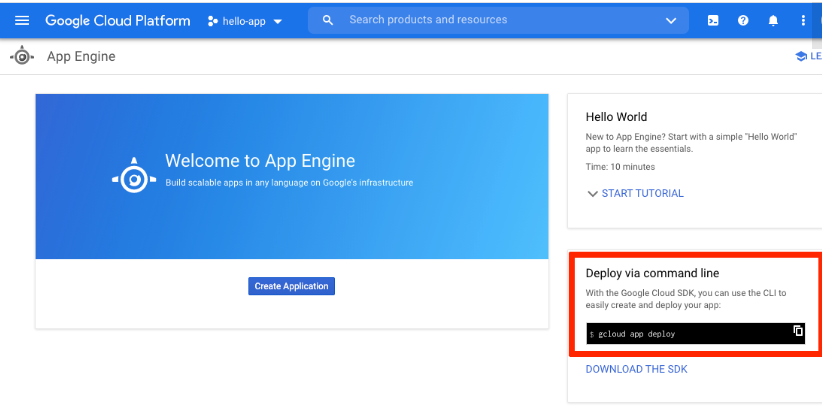
Find the thatswhatshesaid976@gmail.com account.

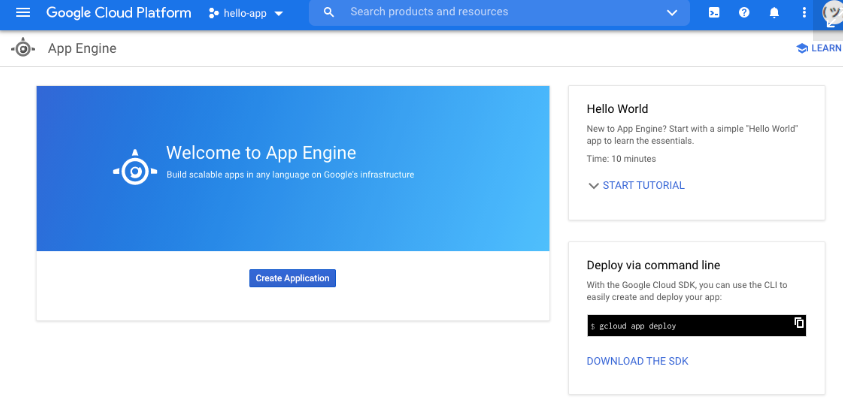
Click "Edit" (pencil icon) next to the account.

Add the App Engine Deployer role (roles/appengine.deployer).

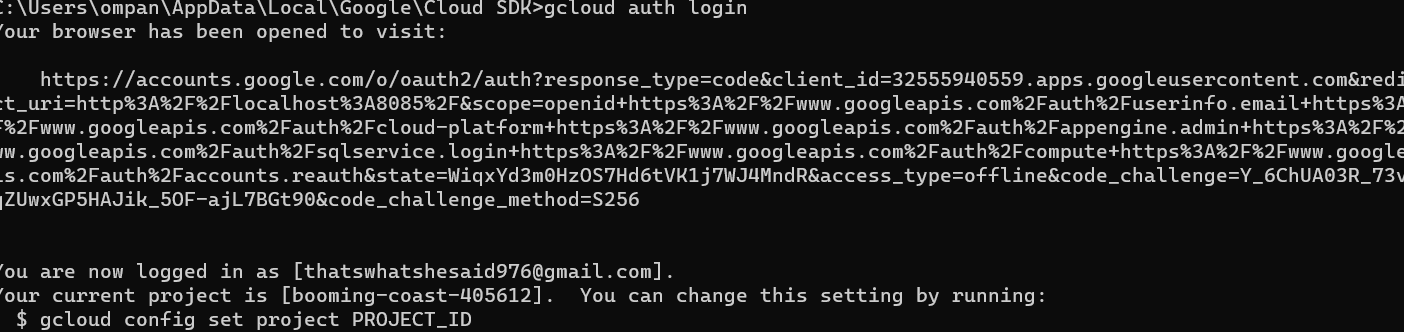
Click "Save."







The gcloud command-line app already provided you with the command that you need to run. Type it into your terminal:



This will start the authentication process by generating a validation URL and opening it up in your browser. Complete the process by selecting your Google account in the browser window and granting Google Cloud SDK the necessary privileges. After you do this, you can return to your terminal, where you’ll see some information about the authentication process: