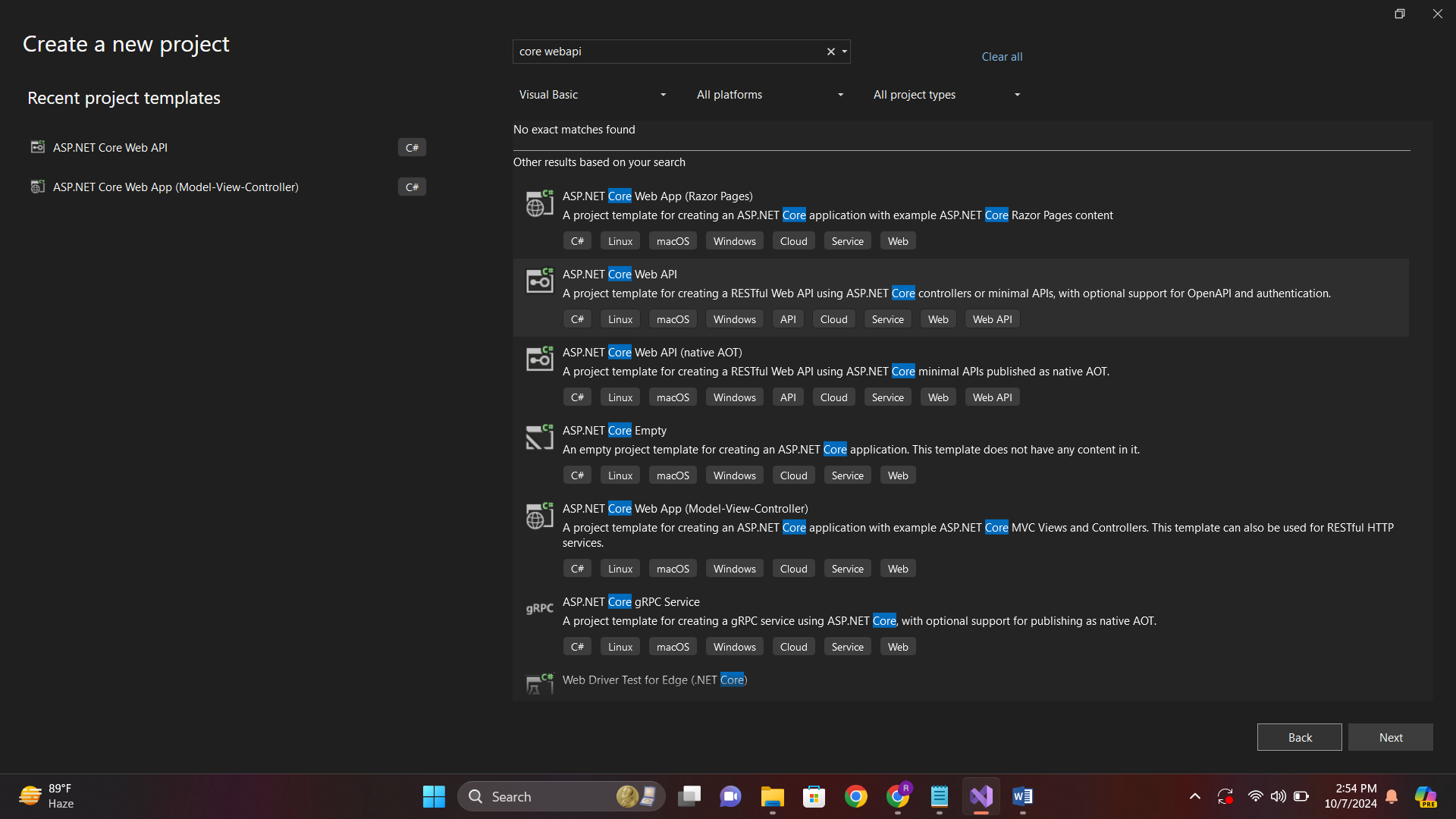
**PROJECT : STUDENT TEST**

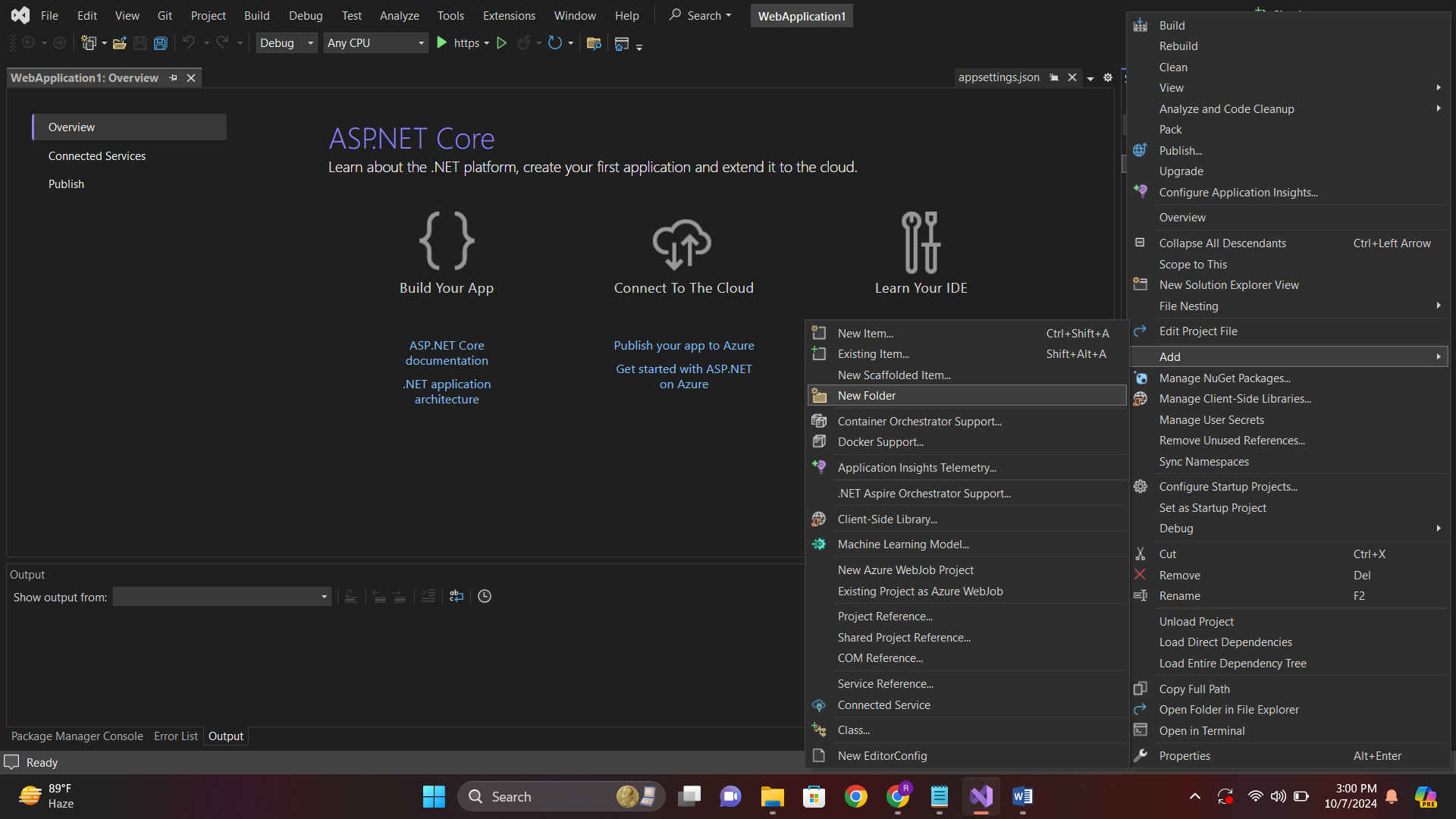
**TECHNOLOGY: WEB API CORE DATABASE FIRST APPROACH**

**Project Overview:**

* **Technologies**: ASP.NET Web API Core (back-end),C#, HTML (front-end), JavaScript (front-end logic).
* **Features**:
  1. **Student Registration**: A form for students to register with Username, Mail and Password.
  2. **Login**: Students log in using their credentials.
  3. **Test Interface**: After login, students are presented with a set of test questions.
  4. **Test Submission**: Students submit answers, and the score is calculated.

**Step1 :** create ASP.NET CORE WEB API project



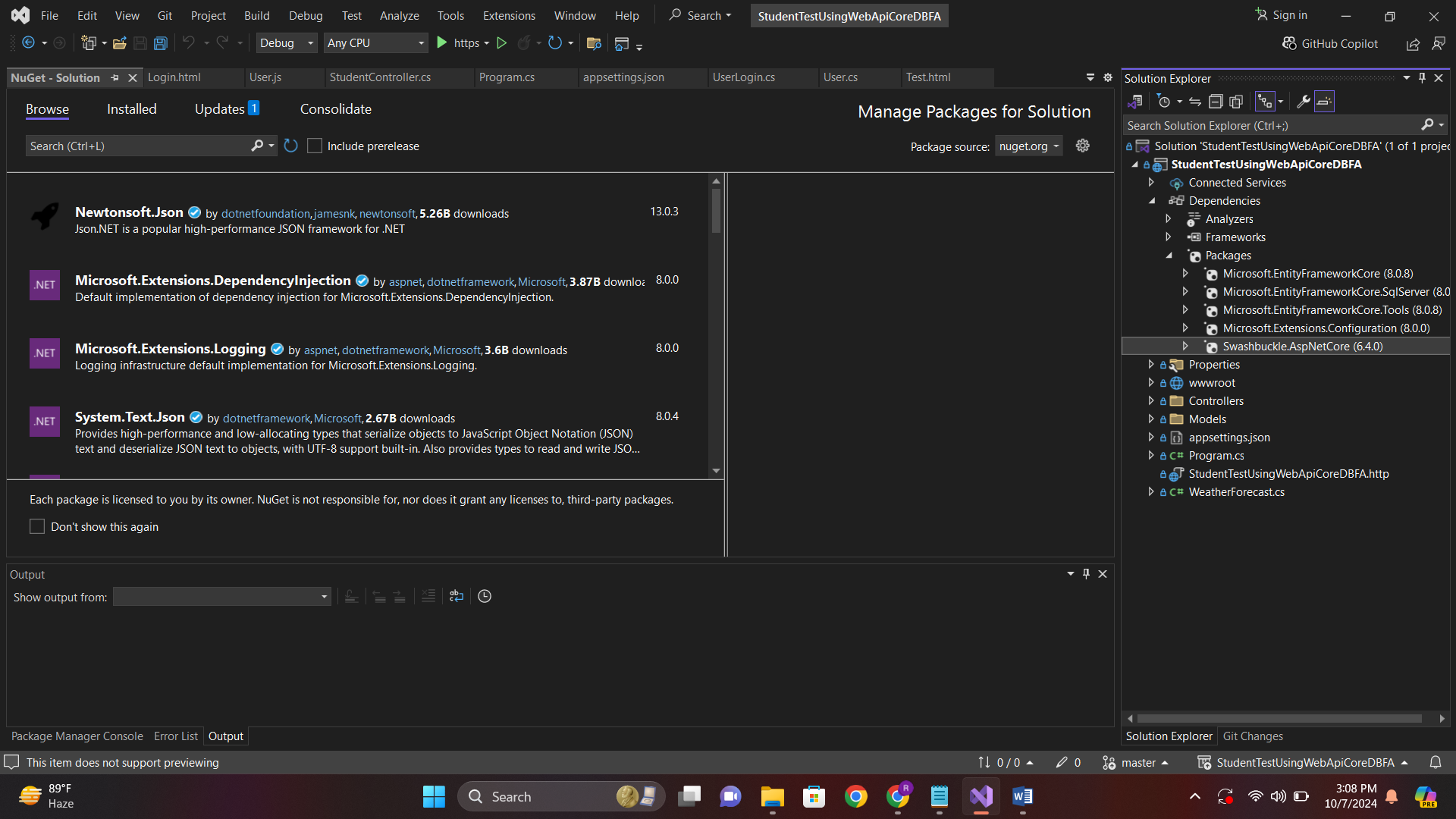
**Step 2:** after creating project add **Models** and **wwwroot** folders.

**Step 3:** add the necessary packages

1. Microsoft.EntityFrameworkCore

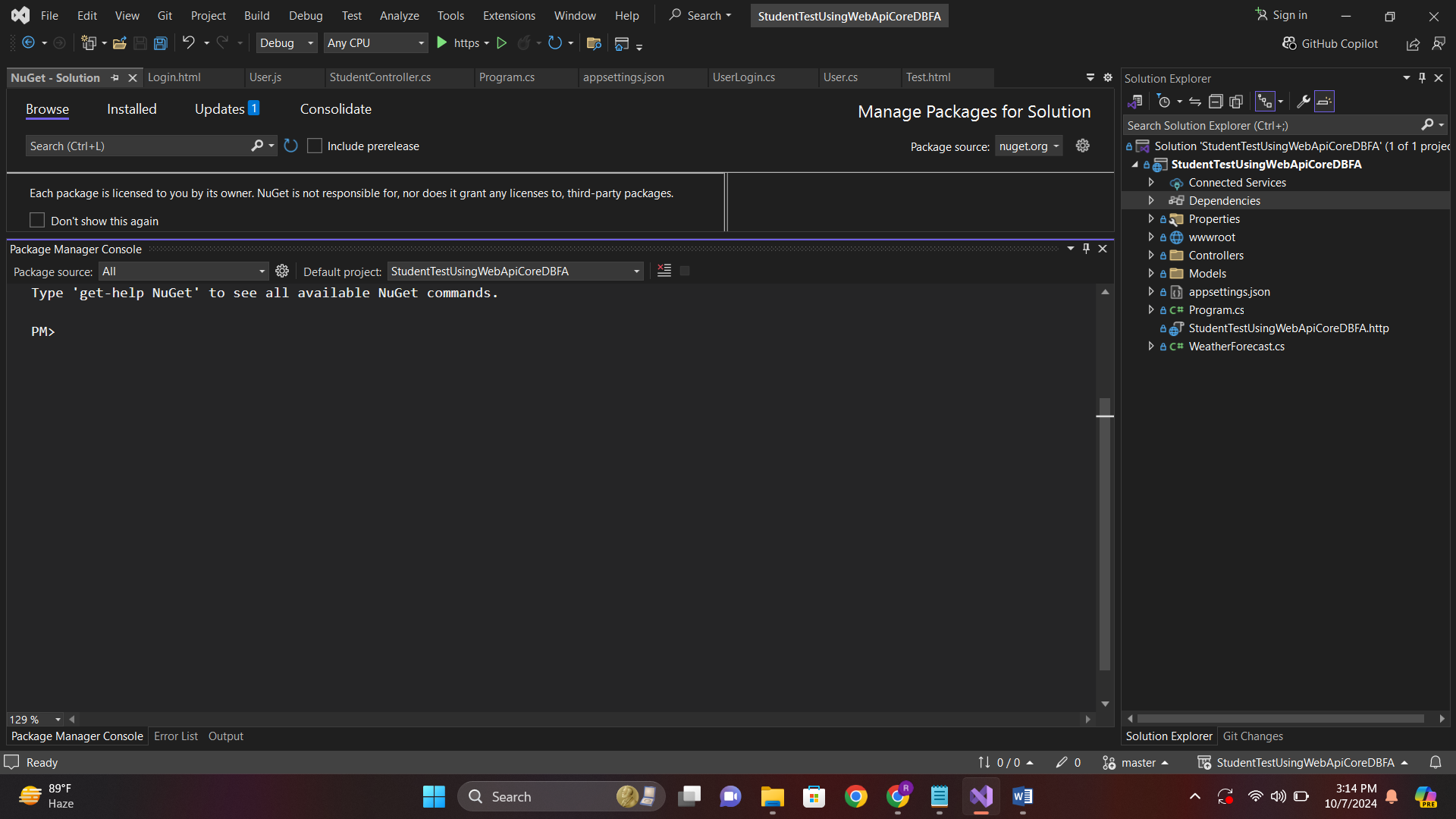
2. Microsoft.EntityFrameworkCore.SqlServer

3. Microsoft.EntityFrameworkCore.Tools

4. Microsoft.Extension.Configurations**Step 4:** goto->Tool->Nuget Package Manager->Package Manager Conseole.

Type This Command: Scaffold-DbContext "server=AHMAD\SQLEXPRESS;Database=DbForCrud;Trusted\_Connection=True;TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

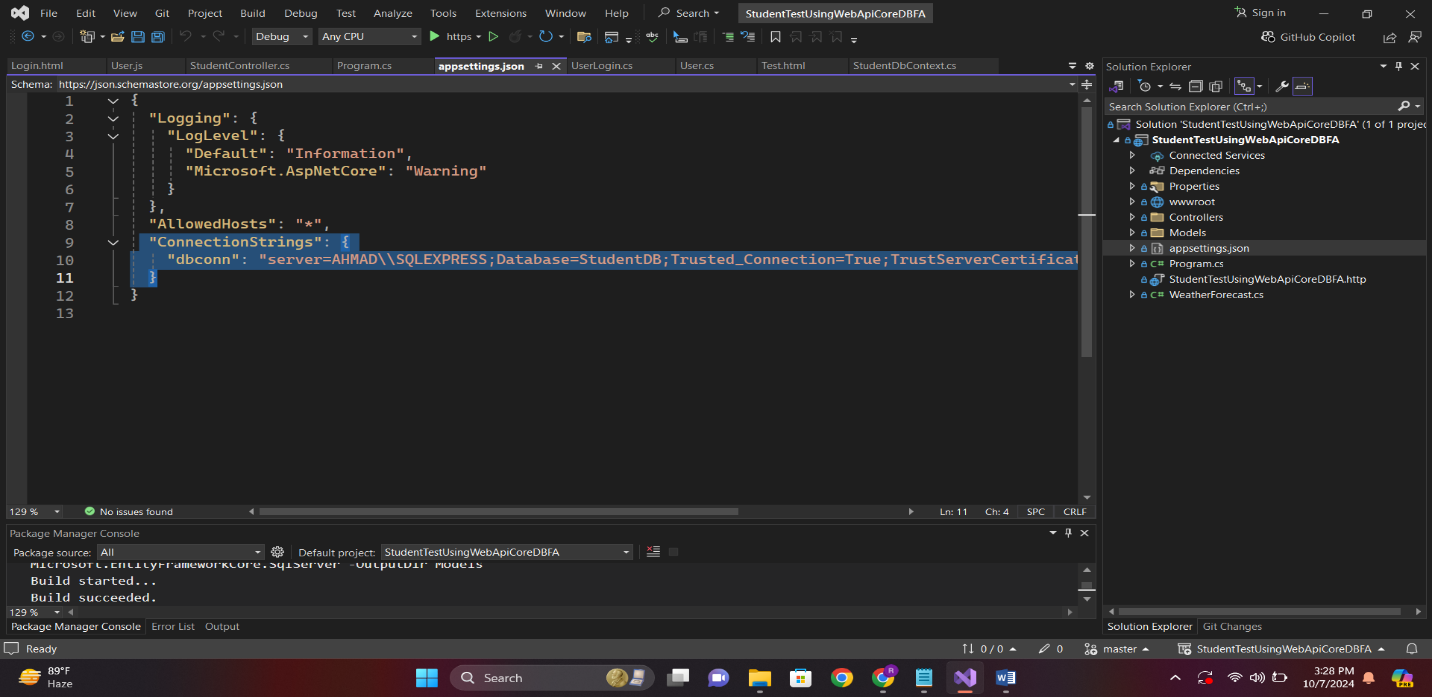
**(Note: Model classes and DbContext automatically created in Model Folder)**



**Step 5:** goto Appsetting.json ,give a connection string.

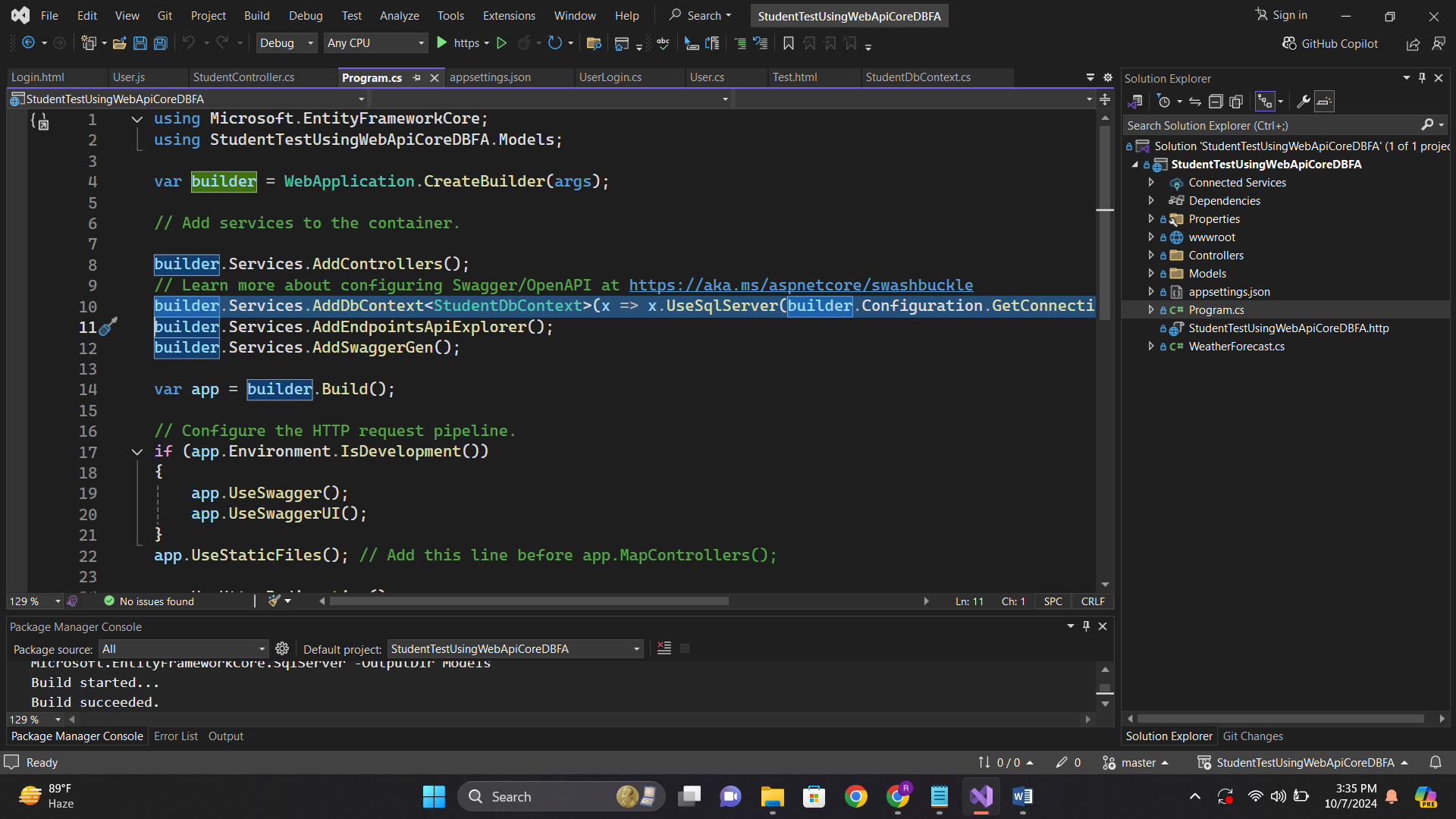
"ConnectionStrings": {

"dbconn": "server=AHMAD\\SQLEXPRESS;Database=StudentDB;Trusted\_Connection=True;TrustServerCertificate=True;"

}

**Step 6:** goto Program.cs and build service.

builder.Services.AddDbContext<StudentDbContext>(x => x.UseSqlServer(builder.Configuration.GetConnectionString("dbconn")));



**Step 7:** goto controller and create studentcontroller and build logic like below.

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using StudentTestUsingWebApiCoreDBFA.Models;

namespace StudentTestUsingWebApiCoreDBFA.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class StudentController : ControllerBase

{

private readonly StudentDbContext \_context;

public StudentController(StudentDbContext context)

{

\_context = context;

}

// POST: api/user/register

[HttpPost("register")]

public IActionResult Register([FromBody] User model)

{

if (\_context.Users.Any(u => u.UserName == model.UserName || u.Email == model.Email))

{

return BadRequest("Username or Email already exists.");

}

\_context.Users.Add(model);

\_context.SaveChanges();

return Ok();

}

// POST: api/user/login

[HttpPost("login")]

public IActionResult Login([FromBody] LoginRequest model)

{

if (model == null || string.IsNullOrEmpty(model.UserName) || string.IsNullOrEmpty(model.Password))

{

return BadRequest("Username and password are required.");

}

var user = \_context.Users.FirstOrDefault(u => u.UserName == model.UserName && u.Password == model.Password);

if (user == null)

{

return Unauthorized("Invalid username or password.");

}

var loginRecord = new UserLogin

{

UserId = user.UserId,

LoginTime = DateTime.Now,

IsLoggedIn = true

};

\_context.UserLogins.Add(loginRecord);

\_context.SaveChangesAsync();

return Ok();

}

// GET: api/user/start-test

[HttpGet("startTest")]

public IActionResult StartTest()

{

var questions = \_context.Questions

.Select(q => new

{

q.QuestionId,

q.QuestionText,

q.Explanation,

Answers = q.Answers.Select(a => new

{

a.AnswerId,

a.AnswerText

}).ToList()

})

.OrderBy(q => Guid.NewGuid())

.Take(5)

.ToList();

return Ok(questions);

}

// POST: api/user/submit-test

[HttpPost("submit-test")]

public IActionResult SubmitTest([FromBody] List<TestSubmission> answers)

{

int correctAnswers = 0;

foreach (var submission in answers)

{

var correctAnswer = \_context.Answers

.FirstOrDefault(a => a.QuestionId == submission.QuestionId && a.IsCorrect);

if (correctAnswer != null && correctAnswer.AnswerId == submission.SelectedAnswerId)

{

correctAnswers++;

}

}

return Ok(new { Score = correctAnswers });

}

// GET: api/user/result/{score}

[HttpGet("result/{score}")]

public IActionResult Result(int score)

{

int totalQuestions = \_context.Answers.Where(a => a.IsCorrect).Count();

double percentageScore = ((double)score / totalQuestions) \* 100;

return Ok(new

{

Score = score,

TotalQuestions = totalQuestions,

PercentageScore = percentageScore

});

}

}

}

**Step 8:** goto **wwwroot**-> add->NewItem->JavaScriptFile

// wwwroot/js/user.js

async function registerUser() {

const user = {

userName: document.getElementById("username").value,

email: document.getElementById("email").value,

password: document.getElementById("password").value

};

const response = await fetch('/api/Student/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(user)

});

if (response.ok) {

alert('Registration successful');

window.location.href = '/login.html';

} else {

alert('Registration failed: ' + await response.text());

}

}

async function loginUser() {

const loginRequest = {

userName: document.getElementById("uname").value,

password: document.getElementById("pass").value

};

const response = await fetch('/api/Student/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(loginRequest)

});

if (response.ok) {

window.location.href = '/test.html';

} else {

alert('Login failed: ' + await response.text());

}

}

async function startTest() {

try {

const response = await fetch('/api/Student/starTtest');

if (!response.ok) {

throw new Error('Failed to fetch questions: ' + response.statusText);

}

const questions = await response.json();

if (questions.length ===0) {

document.getElementById('questions').innerHTML = '<p>No questions available.</p>';

return;

}

let questionHtml = '';

// Check if questions are coming in a valid array format

if (Array.isArray(questions) && questions.length > 0) {

debugger;

questions.forEach((q, index) => {

questionHtml += `<div class="question">

<p>${index + 1}. ${q.QuestionText}</p>

${q.Answers.map(a => `

<label>

<input type="radio" name="q${q.QuestionId}" value="${a.AnswerId}">

${a.AnswerText}

</label><br>`).join('')}

</div>`;

});

} else {

questionHtml = `<p>No questions available.</p>`;

}

document.getElementById('questions').innerHTML = questionHtml;

} catch (error) {

console.error('Error loading test questions:', error);

document.getElementById('questions').innerHTML = '<p>Error loading test questions.</p>';

}

}

async function submitTest() {

const answers = [];

document.querySelectorAll('.question').forEach(questionDiv => {

const questionId = questionDiv.querySelector('input[type="radio"]').name.slice(1);

const selectedAnswerId = questionDiv.querySelector('input[type="radio"]:checked')?.value;

if (selectedAnswerId) {

answers.push({ QuestionId: questionId, SelectedAnswerId: selectedAnswerId });

}

});

const response = await fetch('/api/Student/submit-test', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(answers)

});

const result = await response.json();

window.location.href = `/pages/result.html?score=${result.Score}`;

}

async function showResult() {

const params = new URLSearchParams(window.location.search);

const score = params.get('score');

const response = await fetch(`/api/user/result/${score}`);

const result = await response.json();

document.getElementById('result').innerHTML = `

<p>Your score: ${result.Score}</p>

<p>Total questions: ${result.TotalQuestions}</p>

<p>Percentage: ${result.PercentageScore.toFixed(2)}%</p>

`;

}

**Explanation For User.Js File** :

1. **registerUser()**:
   * Captures user input (username, email, password) from form fields.
   * Sends a POST request to the /api/Student/register endpoint, with the data in JSON format.
   * If registration is successful, redirects the user to the login page. Otherwise, displays an error message.
2. **loginUser()**:
   * Collects the username and password from the login form.
   * Sends a POST request to /api/Student/login with the login credentials.
   * On success, redirects the user to the test page. On failure, shows an error message.
3. **startTest()**:
   * Sends a request to /api/Student/startTest to fetch the test questions.
   * If successful, it generates HTML to display each question with multiple-choice answers. If no questions are available, it informs the user.
4. **submitTest()**:
   * Collects the selected answers from the test page.
   * Sends a POST request to /api/Student/submit-test, sending the selected answers in JSON format.
   * Redirects the user to the result page with the test score as a query parameter.
5. **showResult()**:
   * Extracts the score from the URL query string.
   * Sends a request to /api/user/result/{score} to fetch and display the test results (score, total questions, percentage).

Each method communicates with the back-end Web API using fetch and processes the server responses for user interaction.

Note : If you want to create Html pages for Register,Login,Test Youn can create your own.