**✅ PHASE 1: DESIGN DOCUMENTS & PLANNING (for Presentation)**

**📌 1. Database Schema Design (ER Diagram Description)**

We’ll create **3 core tables**:

**🔹 Table: Users**

| **Column Name** | **Data Type** | **Constraints** |
| --- | --- | --- |
| UserId | INT | Primary Key, Identity |
| Name | NVARCHAR(100) | NOT NULL |
| Email | NVARCHAR(100) | UNIQUE, NOT NULL |
| PasswordHash | NVARCHAR(255) | NOT NULL |
| Bio | NVARCHAR(255) | NULL |
| CreatedAt | DATETIME | Default: GETDATE() |

**🔹 Table: Recipes**

| **Column Name** | **Data Type** | **Constraints** |
| --- | --- | --- |
| RecipeId | INT | Primary Key, Identity |
| Title | NVARCHAR(100) | NOT NULL |
| Description | TEXT | NULL |
| IsVegetarian | BIT | NOT NULL |
| Region | NVARCHAR(100) | NULL |
| CreatedAt | DATETIME | Default: GETDATE() |
| UserId | INT | Foreign Key → Users |

**🔹 Table: Feedback**

| **Column Name** | **Data Type** | **Constraints** |
| --- | --- | --- |
| FeedbackId | INT | Primary Key, Identity |
| Rating | INT | CHECK (Rating BETWEEN 1 AND 5) |
| Comment | TEXT | NULL |
| CreatedAt | DATETIME | Default: GETDATE() |
| UserId | INT | Foreign Key → Users |
| RecipeId | INT | Foreign Key → Recipes |

**🎯 Relationships**

* One User → Many Recipes
* One User → Many Feedback
* One Recipe → Many Feedback

**🗂 SQL Script to Create the DB**

Would you like the SQL script now? I can generate it for direct use in SSMS.

**🧾 2. Use Case Diagram**

**Actors:**

* User (Registered/Guest)

**Use Cases:**

* Register
* Login
* Update Profile
* Add Recipe
* View Recipes
* Filter Recipes
* Delete Own Recipe
* Add Feedback (Rating + Comment)
* View Feedback

I will generate a use case diagram (visual format). Want it in text, draw.io, or image form?

**🔁 3. Data Flow Diagram (DFD – Level 0 & Level 1)**

**DFD Level 0:**

scss

CopyEdit

[User] ---> (Online Recipe Portal System) ---> [Database]

**DFD Level 1:**

Breaks into:

* Login/Register → Users Table
* Add/View Recipe → Recipes Table
* Add/View Feedback → Feedback Table

(Visual diagram can be created)

**📦 4. Class Diagram (Backend Model)**

**Classes:**

csharp

CopyEdit

class User {

int UserId;

string Name;

string Email;

string PasswordHash;

string Bio;

DateTime CreatedAt;

List<Recipe> Recipes;

List<Feedback> Feedbacks;

}

class Recipe {

int RecipeId;

string Title;

string Description;

bool IsVegetarian;

string Region;

DateTime CreatedAt;

int UserId;

User User;

List<Feedback> Feedbacks;

}

class Feedback {

int FeedbackId;

int Rating;

string Comment;

DateTime CreatedAt;

int UserId;

int RecipeId;

User User;

Recipe Recipe;

}

**🧱 5. High-Level Design (HLD)**

**🔷 Objective:**

Design an online platform to share, view, and rate cooking recipes categorized by type and region.

**🔷 Components:**

* **Frontend (Angular)**  
  Login/Register, Recipe List, Add Recipe, Profile, Feedback
* **Backend (ASP.NET Core Web API)**  
  Auth Controller, Recipe Controller, Feedback Controller
* **Database**  
  Users, Recipes, Feedback

**🔷 Architecture:**

* RESTful APIs with JWT Auth
* Angular communicates with backend via HTTP calls
* SQL Server DB accessed using EF Core (Database First)

**🔷 Users:**

* End users can register, add/view/delete recipes, and give feedback.

**🧰 6. Low-Level Design (LLD)**

**📁 Backend Folder Structure**

lua

CopyEdit

CookingPortalAPI/

│

├── Controllers/

│ ├── AuthController.cs

│ ├── RecipeController.cs

│ └── FeedbackController.cs

│

├── Models/ <-- Auto-generated from DB

│

├── DTOs/

│ ├── RegisterDto.cs

│ ├── LoginDto.cs

│ ├── RecipeDto.cs

│ └── FeedbackDto.cs

│

├── Services/

│ ├── IAuthService.cs / AuthService.cs

│ ├── IRecipeService.cs / RecipeService.cs

│ └── IFeedbackService.cs / FeedbackService.cs

│

├── Data/

│ └── CookingDbContext.cs <-- Scaffolded

│

├── Middleware/

│ └── JWTAuthMiddleware.cs

**🔄 Sample API Contracts**

* POST /api/auth/register → Register user
* POST /api/auth/login → Returns JWT
* GET /api/recipes → List all
* POST /api/recipes → Add recipe
* POST /api/feedback → Add feedback

Here are the **SQL Server commands** to create the database and the three tables (Users, Recipes, Feedback) for your **Online Cooking Recipe Portal System**, including relationships and constraints.

CREATE DATABASE CookingRecipePortalDB;

GO

USE CookingRecipePortalDB;

GO

CREATE TABLE Users (

UserId INT PRIMARY KEY IDENTITY(1,1),

Name NVARCHAR(100) NOT NULL,

Email NVARCHAR(100) NOT NULL UNIQUE,

PasswordHash NVARCHAR(255) NOT NULL,

Bio NVARCHAR(255),

CreatedAt DATETIME DEFAULT GETDATE()

);

CREATE TABLE Recipes (

RecipeId INT PRIMARY KEY IDENTITY(1,1),

Title NVARCHAR(100) NOT NULL,

Description TEXT,

IsVegetarian BIT NOT NULL,

Region NVARCHAR(100),

CreatedAt DATETIME DEFAULT GETDATE(),

UserId INT NOT NULL,

CONSTRAINT FK\_Recipes\_Users FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE

);

CREATE TABLE Feedback (

FeedbackId INT PRIMARY KEY IDENTITY(1,1),

Rating INT CHECK (Rating BETWEEN 1 AND 5),

Comment TEXT,

CreatedAt DATETIME DEFAULT GETDATE(),

UserId INT NOT NULL,

RecipeId INT NOT NULL,

CONSTRAINT FK\_Feedback\_Users FOREIGN KEY (UserId) REFERENCES Users(UserId) ON DELETE CASCADE,

CONSTRAINT FK\_Feedback\_Recipes FOREIGN KEY (RecipeId) REFERENCES Recipes(RecipeId) ON DELETE CASCADE

);

-- Insert sample users

INSERT INTO Users (Name, Email, PasswordHash, Bio)

VALUES

('Priya', 'priya@example.com', 'hashed\_password\_1', 'South Indian cooking expert'),

('Ravi', 'ravi@example.com', 'hashed\_password\_2', NULL);

-- Insert sample recipe

INSERT INTO Recipes (Title, Description, IsVegetarian, Region, UserId)

VALUES

('Bisi Bele Bath', 'Traditional Karnataka rice dish', 1, 'Karnataka', 1);

-- Insert feedback

INSERT INTO Feedback (Rating, Comment, UserId, RecipeId)

VALUES

(5, 'Delicious and easy to make!', 2, 1);

**✅ PHASE 2: Scaffold Backend with Database First**

**✅ Step 1: Create ASP.NET Core Web API Project**

1. Open **Visual Studio 2022**
2. Click **Create a new project**
3. Choose **ASP.NET Core Web API**
4. Click **Next**, name it CookingRecipePortalAPI
5. Select:
   * Framework: **.NET 7 or 8**
   * Uncheck **HTTPS** if testing locally
   * Enable **Controllers**, NOT minimal API
6. Click **Create**

**✅ Step 2: Install Required NuGet Packages**

Open **Package Manager Console (PMC)** and run:

powershell

CopyEdit

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Install-Package Microsoft.EntityFrameworkCore.Design

**✅ Step 3: Scaffold the Models from the Database**

In **PMC**, run this command (adjust connection string if needed):

powershell

CopyEdit

Scaffold-DbContext "Server=(local)\SQLEXPRESS;Database=CookingRecipePortalDB;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context CookingDbContext -DataAnnotations -Force

🔹 This will:

* Create CookingDbContext.cs
* Create 3 entity classes: User, Recipe, Feedback
* Place them in the Models folder

**✅ Step 4: Verify the Generated Code**

Ensure:

* **CookingDbContext.cs** has DbSet<User>, DbSet<Recipe>, etc.
* Relationships like User.Recipes, Recipe.Feedbacks, etc., are generated.

**✅ Step 5: Register DbContext in Program.cs**

In Program.cs, add:

csharp

CopyEdit

builder.Services.AddDbContext<CookingDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

In appsettings.json, add your connection string:

json

CopyEdit

"ConnectionStrings": {

"DefaultConnection": "Server=(local)\\SQLEXPRESS;Database=CookingRecipePortalDB;Trusted\_Connection=True;"

}

**✅ Step 6: Create Folder Structure**

Organize your solution like this:

lua

CopyEdit

CookingRecipePortalAPI/

├── Controllers/

│ ├── AuthController.cs

│ ├── RecipeController.cs

│ └── FeedbackController.cs

├── DTOs/

│ ├── RegisterDto.cs

│ ├── LoginDto.cs

│ └── RecipeDto.cs (etc.)

├── Models/ <-- Already generated

├── Services/

│ ├── Interfaces/

│ └── Implementations/

└── CookingDbContext.cs

**✅ PHASE 3: Auth Functionality (Backend)**

**✅ Step 1: Add JWT Support – Install Package**

In **Package Manager Console** run:

powershell

CopyEdit

Install-Package Microsoft.AspNetCore.Authentication.JwtBearer

**✅ Step 2: Add JWT Settings to appsettings.json**

json

CopyEdit

"JwtSettings": {

"Key": "ThisIsASecretKeyForJwt1234567890", // Minimum 16 characters

"Issuer": "CookingApp",

"Audience": "CookingAppUser",

"ExpiryMinutes": 60

}

**✅ Step 3: Register JWT in Program.cs**

Add these:

csharp

CopyEdit

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var jwtSettings = builder.Configuration.GetSection("JwtSettings");

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings["Issuer"],

ValidAudience = jwtSettings["Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["Key"]))

};

});

builder.Services.AddAuthorization();

And in app.UseAuthentication(); and app.UseAuthorization();

**✅ Step 4: Create DTOs**

📁 DTOs/RegisterDto.cs

csharp

CopyEdit

public class RegisterDto

{

public string Name { get; set; }

public string Email { get; set; }

public string Password { get; set; }

}

📁 DTOs/LoginDto.cs

csharp

CopyEdit

public class LoginDto

{

public string Email { get; set; }

public string Password { get; set; }

}

**✅ Step 5: Create AuthController**

📁 Controllers/AuthController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using CookingRecipePortalAPI.Models;

using CookingRecipePortalAPI.DTOs;

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly CookingDbContext \_context;

private readonly IConfiguration \_config;

public AuthController(CookingDbContext context, IConfiguration config)

{

\_context = context;

\_config = config;

}

[HttpPost("register")]

public IActionResult Register(RegisterDto dto)

{

if (\_context.Users.Any(u => u.Email == dto.Email))

return BadRequest("Email already exists.");

var user = new User

{

Name = dto.Name,

Email = dto.Email,

PasswordHash = BCrypt.Net.BCrypt.HashPassword(dto.Password)

};

\_context.Users.Add(user);

\_context.SaveChanges();

return Ok("User registered successfully.");

}

[HttpPost("login")]

public IActionResult Login(LoginDto dto)

{

var user = \_context.Users.FirstOrDefault(u => u.Email == dto.Email);

if (user == null || !BCrypt.Net.BCrypt.Verify(dto.Password, user.PasswordHash))

return Unauthorized("Invalid credentials.");

var token = GenerateToken(user);

return Ok(new { Token = token, UserId = user.UserId, Name = user.Name });

}

private string GenerateToken(User user)

{

var jwtSettings = \_config.GetSection("JwtSettings");

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.Name, user.Name)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["Key"]));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: jwtSettings["Issuer"],

audience: jwtSettings["Audience"],

claims: claims,

expires: DateTime.Now.AddMinutes(Convert.ToDouble(jwtSettings["ExpiryMinutes"])),

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**✅ Step 6: Test Your APIs**

Use **Postman**:

* **Register**: POST http://localhost:xxxx/api/auth/register

json

CopyEdit

{

"name": "Priya",

"email": "priya@example.com",

"password": "test123"

}

* **Login**: POST http://localhost:xxxx/api/auth/login

json

CopyEdit

{

"email": "priya@example.com",

"password": "test123"

}

It should return a JWT token.

**🔐 Next Steps:**

Would you like me to:

1. Build the **RecipeController** (Add/View/Delete Recipes)?
2. Build **Profile update (edit bio)** feature?
3. Or secure the routes with [Authorize]?

Let me know — we’ll continue with the next precise module.

**You said:**

Build the RecipeController (Add/View/Delete Recipes)? Build Profile update (edit bio) feature? Or secure the routes with [Authorize]?

**ChatGPT said:**

Great! We'll do **all three** — in the right order — so everything works smoothly and securely. Here's the plan:

**✅ PHASE 4: Build & Secure RecipeController + Profile Update**

**✅ Step 1: Secure the Application with [Authorize]**

**🔒 Apply Authentication Globally or per Controller**

In Program.cs, **ensure you have:**

csharp

CopyEdit

app.UseAuthentication();

app.UseAuthorization();

**✅ In RecipeController, Add:**

csharp

CopyEdit

[Authorize] // Apply to controller or individual methods

✅ You're now protecting APIs that require a logged-in user (with JWT token).

**✅ Step 2: Create RecipeController with Add/View/Delete APIs**

📁 Controllers/RecipeController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using CookingRecipePortalAPI.Models;

using System.Security.Claims;

[ApiController]

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

[Authorize]

public class RecipeController : ControllerBase

{

private readonly CookingDbContext \_context;

public RecipeController(CookingDbContext context)

{

\_context = context;

}

// POST: api/recipe

[HttpPost]

public IActionResult AddRecipe(Recipe recipe)

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

recipe.UserId = userId;

recipe.CreatedAt = DateTime.Now;

\_context.Recipes.Add(recipe);

\_context.SaveChanges();

return Ok(recipe);

}

// GET: api/recipe

[AllowAnonymous] // Public

[HttpGet]

public IActionResult GetAllRecipes()

{

var recipes = \_context.Recipes

.Select(r => new

{

r.RecipeId,

r.Title,

r.Description,

r.IsVegetarian,

r.Region,

r.CreatedAt,

UserName = r.User.Name

})

.ToList();

return Ok(recipes);

}

// GET: api/recipe/filter?isVegetarian=true&region=Karnataka

[AllowAnonymous]

[HttpGet("filter")]

public IActionResult FilterRecipes(bool? isVegetarian, string? region)

{

var query = \_context.Recipes.AsQueryable();

if (isVegetarian != null)

query = query.Where(r => r.IsVegetarian == isVegetarian);

if (!string.IsNullOrEmpty(region))

query = query.Where(r => r.Region.Contains(region));

var results = query

.Select(r => new

{

r.RecipeId,

r.Title,

r.Description,

r.IsVegetarian,

r.Region,

r.CreatedAt,

UserName = r.User.Name

})

.ToList();

return Ok(results);

}

// DELETE: api/recipe/{id}

[HttpDelete("{id}")]

public IActionResult DeleteRecipe(int id)

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

var recipe = \_context.Recipes.FirstOrDefault(r => r.RecipeId == id && r.UserId == userId);

if (recipe == null)

return NotFound("Recipe not found or not yours.");

\_context.Recipes.Remove(recipe);

\_context.SaveChanges();

return Ok("Recipe deleted.");

}

}

**✅ Step 3: Create ProfileController for Updating Bio**

📁 Controllers/ProfileController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using CookingRecipePortalAPI.Models;

using System.Security.Claims;

[ApiController]

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

[Authorize]

public class ProfileController : ControllerBase

{

private readonly CookingDbContext \_context;

public ProfileController(CookingDbContext context)

{

\_context = context;

}

// GET: api/profile

[HttpGet]

public IActionResult GetMyProfile()

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

var user = \_context.Users

.Where(u => u.UserId == userId)

.Select(u => new

{

u.UserId,

u.Name,

u.Email,

u.Bio,

u.CreatedAt

}).FirstOrDefault();

return Ok(user);

}

// PUT: api/profile

[HttpPut]

public IActionResult UpdateBio([FromBody] string bio)

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

var user = \_context.Users.FirstOrDefault(u => u.UserId == userId);

if (user == null) return NotFound("User not found.");

user.Bio = bio;

\_context.SaveChanges();

return Ok("Bio updated.");

}

}

**✅ Step 4: Test in Postman (with JWT)**

**1. 🔐 Get JWT from api/auth/login**

Paste token in Postman → **Authorization → Bearer Token**

**2. 🔼 Add Recipe**

http

CopyEdit

POST /api/recipe

{

"title": "Bisi Bele Bath",

"description": "Traditional Karnataka dish",

"isVegetarian": true,

"region": "Karnataka"

}

**3. 👁 View All Recipes**

http

CopyEdit

GET /api/recipe

**4. ❌ Delete Own Recipe**

http

CopyEdit

DELETE /api/recipe/1

**5. 🔁 Get Profile**

http

CopyEdit

GET /api/profile

**6. ✏️ Update Bio**

http

CopyEdit

PUT /api/profile

"Expert in South Indian cuisine"

**✅ Next Steps:**

Would you like me to now:

1. Build the **FeedbackController** (ratings/comments)?
2. Help you set up the **Angular frontend structure**?
3. Or write the **service layer for clean architecture**?

Let me know and we’ll continue step-by-step!

**You said:**

Build the FeedbackController

**ChatGPT said:**

Great, Ashirvad! Let’s now build the complete **FeedbackController** with:

* ✅ Add feedback (rating + comment)
* ✅ View feedbacks for a recipe
* ✅ Authorization (only logged-in users can post feedback)

**✅ PHASE 5: Build FeedbackController**

**✅ Step 1: Feedback DTO**

📁 DTOs/FeedbackDto.cs

csharp

CopyEdit

public class FeedbackDto

{

public int RecipeId { get; set; }

public int Rating { get; set; }

public string? Comment { get; set; }

}

**✅ Step 2: FeedbackController Code**

📁 Controllers/FeedbackController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using CookingRecipePortalAPI.Models;

using CookingRecipePortalAPI.DTOs;

using System.Security.Claims;

[ApiController]

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

[Authorize]

public class FeedbackController : ControllerBase

{

private readonly CookingDbContext \_context;

public FeedbackController(CookingDbContext context)

{

\_context = context;

}

// POST: api/feedback

[HttpPost]

public IActionResult AddFeedback([FromBody] FeedbackDto dto)

{

if (!\_context.Recipes.Any(r => r.RecipeId == dto.RecipeId))

return NotFound("Recipe not found.");

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

var feedback = new Feedback

{

RecipeId = dto.RecipeId,

UserId = userId,

Rating = dto.Rating,

Comment = dto.Comment,

CreatedAt = DateTime.Now

};

\_context.Feedbacks.Add(feedback);

\_context.SaveChanges();

return Ok("Feedback submitted.");

}

// GET: api/feedback/recipe/1

[AllowAnonymous]

[HttpGet("recipe/{recipeId}")]

public IActionResult GetFeedbacksForRecipe(int recipeId)

{

var recipeExists = \_context.Recipes.Any(r => r.RecipeId == recipeId);

if (!recipeExists)

return NotFound("Recipe not found.");

var feedbacks = \_context.Feedbacks

.Where(f => f.RecipeId == recipeId)

.Select(f => new

{

f.FeedbackId,

f.Rating,

f.Comment,

f.CreatedAt,

UserName = f.User.Name

}).ToList();

return Ok(feedbacks);

}

}

**✅ Step 3: Test in Postman**

**🔼 Add Feedback**

http

CopyEdit

POST /api/feedback

Authorization: Bearer <your-jwt-token>

Content-Type: application/json

{

"recipeId": 1,

"rating": 5,

"comment": "Delicious and easy to make!"

}

**👁 View Feedbacks for a Recipe**

h

CopyEdit

GET /api/feedback/recipe/1

Response:

json

CopyEdit

[

{

"feedbackId": 1,

"rating": 5,

"comment": "Delicious and easy to make!",

"createdAt": "2025-06-20T14:12:00",

"userName": "Ravi"

}

]

**✅ FeedbackController Summary**

| **Endpoint** | **Description** |
| --- | --- |
| POST /api/feedback | Add feedback (auth required) |
| GET /api/feedback/recipe/{id} | View feedbacks for a recipe (public) |