**Write Up for What’s Cooking App**

1. **Database Creation:**

1) Database Creation: I started by creating a new database named "Rllproject" and switched to it using the USE statement.

2) Table Creation: I created three tables within the "Rllproject" database:

● Admin table to store administrative user information.

● Users table to store user information.

● Recipes table to store information about various recipes.

3) Admin Data Insertion: I inserted two records into the Admin table with administrator names, emails, and passwords.

4) Recipe Data Insertion: I inserted ten records into the Recipes table, each with information about a different recipe. This included fields like name, category, ingredients, description, and submission date. Additionally, I updated the ImageURL for each recipe to include images of the respective dishes.

===============================================================

**2.API creation:**

Step 1: Setting up the ASP.NET Web API Project

1. Open Visual Studio Code (VSCode).

2. Create a new ASP.NET Web API project named "CookingAppAPI." Ensure to uncheck the "configure for HTTPS" option.

3. In the Solution Explorer, navigate to Dependencies and install Entity Framework Tools (`ef tools`) and Entity Framework SQL Server.

4. Build the solution.

Step 2: Scaffold Database Context and Controllers

1. Open the NuGet Package Manager Console in VSCode.

2. Run the following command to scaffold the database context and models for each tables:

Scaffold-DbContext "server=LAPTOP-EA5C4MP1;database=Rllproject;trusted\_connection=true;TrustServerCertificate=true;" Microsoft.EntityFrameworkCore.SqlServer -o Models

3. Add controllers for each tables using the "API Controller with actions using Entity Framework" option.

Step 3: Configure Connection String

1. Open the "RllprojectContext" file and copy the server name from the `OnConfiguring` method.

2. Paste the server name into the `appsettings.json` file under the "ConnectionStrings" section. Ensure to include a comma:

json

"AllowedHosts": "\*",

"ConnectionStrings": {

"BookConStr": "server=LAPTOP-EA5C4MP1;database=Rllproject;trusted\_connection=true;TrustServerCertificate=true;"

}

Step 4: Configure Database Context in Program.cs

1. Open the "Program.cs" file.

2. Add services to configure the "RllprojectContext" with the connection string from "appsettings.json":

csharp

builder.Services.AddDbContext<RllprojectContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("CookingApp")));

==================================================================

**3.MVC Creation:**

1. Created Mvc project and done connection between API and MVC.

2. Done Authentication and Authorization.

3.Ado Connection to do CRUD operation b/w sql data and Mvc.

=================================================================

**4. Login page :**

step1: I create a model class for login in the with the name of login

ste2: This class is likely intended to represent the data associated with a user login form.

step3: Inside the Login class, there are two properties defined: Username and Password. These properties are marked with the [Required] attribute

step4:Next right click on controller then click add then choose controller in that select MVC controller Empty.

step5:In the login controller write a HTTP GET method that returns a View for the login page. It's used when a user navigates to the login page.

step6:write a HTTP POST method that handles the login form submission.

step7:Inside the Login method, the following happens:

1.It checks if the model state is valid, meaning that the submitted data meets the validation rules defined in the Login model.

2.It opens a connection to a SQL Server database using the \_connectionString field.

3.It checks if the submitted username and password are valid for an admin user using the IsValidAdminUser method

4.If the submitted credentials are not valid for either admin or regular users, it adds a model error indicating an "Invalid login attempt" to the ModelState.

5.In that i created the action for registration.

step8:Then Right click on the action login inside the controller then select view then Razore view in that create empty without model .

step9:In the Iused the below steps

1.<div class="wrapper">: This <div> element creates a container with the class "wrapper" to hold the entire login form. This class is used to apply styling to the form container.

2.<div class="row">: This <div> element is used to create a row within the form container. It's common to use rows and columns to structure forms in ASP.NET Core MVC views.

3.<form asp-action="Login">: This <form> element defines an HTML form that will be used for user login. The asp-action attribute specifies the action method (Login) that should handle the form submission when the user clicks the submit button.

4.<div asp-validation-summary="ModelOnly" class="text-danger"></div>: This <div> element is used to display model-level validation errors. If there are any validation errors associated with the model, they will be displayed here in red text.

5.<div class="form-group">: This <div> element represents a form group for the "Username" input field.

same like password filed also did .

6.<div class="form-group button">: This <div> element represents a form group for the submit button.

7.<input type="submit" value="Submit" class="btn btn-primary" />: This <input> element is the submit button for the form. It has the value "Submit" and the classes "btn" and "btn-primary" for styling.

step10:Then I open the layout.cs html in that I gave the class nav item forthis login one .

logout and username navbars:

Step1: in the logreg controller i added the action for this logout .

step2:Then add view for this logout one .

step3:Then in the layout.cs html give the nav class item for this logout one and then give the styles forthis one.

step4:Then in the layout.cs html givethe nav class for this username aslo this code generates a user menu inside a <div>. If the user is authenticated, it displays a greeting message like "Hello, [user's name]" where [user's name] is replaced with the actual name of the authenticated user.

**5.Registeration page:**

Firstly I created a model class for registration and added validations.

Then created a empty controller and added functionality for registration.

Then I created a separate layout in shared folder.

And then used that layout while creating a view for registration controller.

And lastly I did stylings for the page.

**6.Recipe--->Details.cshtml view**

In this view, which is used to display the details of a specific recipe, we begin by defining the model it expects to receive, which is of type `CookingAppMVC.Models.Recipe`. The content of this view is enclosed within an HTML structure.

Styling and Background

- To enhance the visual appeal, the background image is set to a picture of a delicious-looking meal, creating an attractive backdrop.

- The container for the recipe details is styled with a semi-transparent white background, rounded corners, and padding to ensure the content is easily readable.

Recipe Details Table

- A table is used to organize the recipe details, with headers for "Item" and "Detail."

- Various recipe details such as "Name," "Category," "Submission Date," "Ingredients," and "Description" are displayed within the table rows.

Recipe Image

- An image of the recipe, provided as `ViewBag.ImageURL`, is displayed on the right side of the container.

- The image is styled to maintain its aspect ratio, fit within the available space, and have a subtle shadow.

Action Buttons

- Beneath the table, two action buttons are provided: "Edit" and "Delete."

- These buttons are linked to the corresponding actions in the `RecipeController`, allowing users to edit or delete the displayed recipe.

**7. `Details` Action in `RecipeController.cs`**

This action in the `RecipeController` is responsible for handling requests to view the details of a specific recipe. It performs the following actions:

- It initializes an empty `Recipe` object to store the retrieved recipe details.

- An HTTP GET request is made to the API endpoint for the specified recipe, identified by its unique `id`.

- If the response from the API is successful (HTTP status code 200), the received JSON data is deserialized into the `Recipe` object, populating its properties.

- The `ViewBag` is used to store additional information such as the recipe's category and submission date, which are displayed alongside the basic recipe details.

- In case of an unsuccessful API response, an error message is stored in the `ViewBag.ErrorMessage` variable to inform the user that the details retrieval has failed.

- Finally, the view is rendered, passing the `Recipe` object containing the details, and the associated view displays these details to the user.

This combination of the `Details.cshtml` view and the `Details` action in the controller provides a user-friendly interface for viewing recipe details and handles potential error scenarios gracefully, enhancing the overall user experience.

**8. Category Vertical bar**

1--->in view of Receipe I have done my part into the index.cshtml. I have given few html and css codes.

**9. HTML Structure:**

Created a grid container <div> with class "grid-view" to hold the content.

Inside the grid container, there is an unordered list <ul> with classes "list-group" and "list-group-flush" to structure the items.

List items <li> within the unordered list, the first one having class "category-header" serving as a header, and the next two containing hyperlinks styled as buttons with the class "rectangle-link."

CSS Styling:

**10.Grid Container Styles:**

Set a fixed width, background color, and positioned it absolutely on the page.

Applied padding and margins for spacing and alignment.

Enabled vertical scrolling with overflow-y: auto.

List Group Styles:

Removed default list styles with list-style-type: none.

Adjusted padding and margins for proper spacing.

List Item Styles:

Made list item backgrounds transparent and removed borders.

Added padding to the list items for spacing.

Category Header Styles:

Set a background color, padding, and centered text alignment.

Used the 'Times New Roman' font and a larger font size.

Link Styles:

Removed underlines from hyperlinks (text-decoration: none).

Set the font color to black and a larger font size.

Aligned text to the center within list items.

Styled links as rectangular buttons with specific padding, background color (#007bff), white text, and rounded corners.

Added a hover effect to change the background color of the buttons when hovered (#0056b3).

**11.Testing:**

Nunit Testing- Created project using Nunit test project ->Added test class->Installed packages->Nunit ->Nunit Test adapter->moq.

**12.Deploymemt-IIS Deployment**

1.Initally we published our Mvc Project in the local folder. Then in the IIS we created a website CookingAppMvc and I have given the port number as 86.Then in the application pool advanced setting we changed the identity into local system.Then we given the authority system in the security of my sql as db admin.