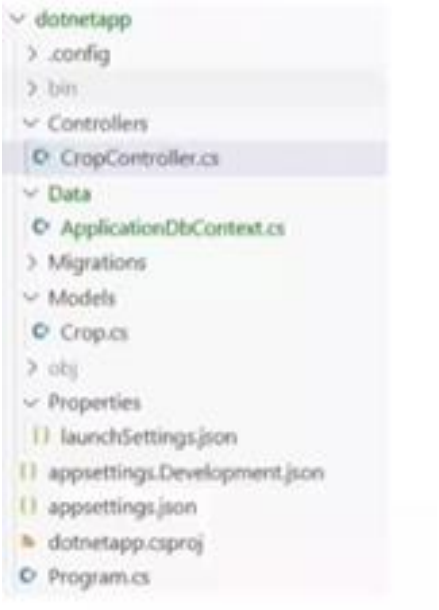
Crop Management System

You need to efficiently store and retrieve crop-related information to streamline agricultural management. This web-based Crop Management System helps users manage crop data by providing a simple interface to track crop types, planting seasons, and yields, ensuring easy access to essential information.

Backend Requirements:

Create folders named as Models, Controllers and configurations inside dotnetapp as mentioned in the below screenshot.



ApplicationDbContext: (/Data/ApplicationDbContext.cs)

Inside Data folder create ApplicationDbContext file with the following DbSet mentioned below

public DbSet<Crop> Crops{ get; set; }

Crop Model (Models/Crop.cs)

This class represents the Crop entity, storing information about different crops, their growth conditions, and yield details. It is used to manage crop data in the system.

Properties:

. CropId (int) -. Unique identifier for the crop.

. Name (string) -. Name of the crop (e.g., Wheat, Rice, Maize).

. Type (string) -. Category of the crop (e.g. Cereal, Grain, Vegetable).

. Season (string) -+ The planting season for the crop (e.g. Summer, Winter)

. HarvestTimeinDays (int) -. Number of days required for the crop to reach harvest.

. Yieldinkg (decimal) -. Expected yield per crop cycle in kilograms.

CropController (Controllers/CropController.cs)

This controller manages crop-related operations, interacting with the ApplicationDbContext to perform CRUD operations on crops.

Functions:

1. public async Task<ActionResult<IEnumerable<Crop>>> GetCrops()

a. Retrieves all crops stored in the database

b. Returns a 200 OK response along with the list of crops.

2. public async Task<ActionResult<Crop>> PostCrop(Crop crop)

a. Adds a new crop to the database.

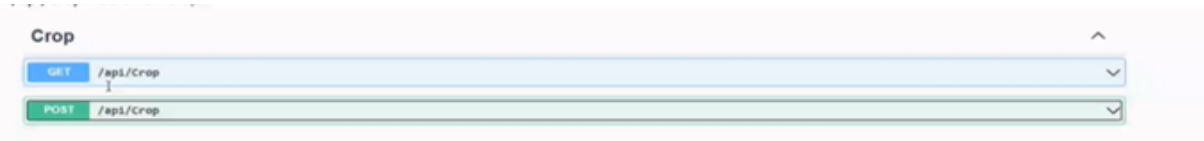
b. Saves changes asynchronously to ensure non-blocking execution.

c. Returns a 200 OK response with a success messago ('Crop added successfully!").

API Endpoints:

1. GET /apl/Crop: Retrieve all crops.

2. POST /api/Crop: Add a new crop.



BACKEND:

Open the terminal and follow the commands below.

. cd dotnetapp

Select the dotnet project folder

. dotnet restore

This command will restore all the required packages to run the application.

· dotnet run

To run the application in port 8080

. dotnet build

To build and check for errors

. dotnet clean

If the same error persists clean the project and build again

To work with Entity Framework Core:

Install EF using the following commands:

dotnet new tool-manifest

dotnet tool install -- local dotnet-ef -- version 6.0.6

dotnet dotnet-ef -- To check the EF installed or not

dotnet dotnet-ef migrations add "InitialSetup" -- command to setup the initial creation of tables mentioned in DBContext

dotnet dotnet-ef database update -- command to update the database

To Work with SQLServer:

(Open a New Terminal) type the below commands

sqlcmd -U sa

password: examlyMssql@123

>use DBName

>go

1> insert into TableName values(" *. "*, ..)

2> go

Note for backend:

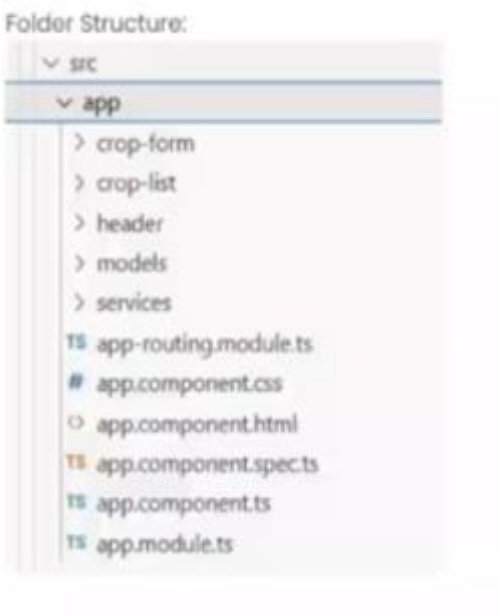
I. Please ensure that the application is running on port 8080 before clicking the "Run Test Case" button.

2. Database Name should be appdb

3. Use the below sample connection string to connect the Ms SQL Server

connectionString = "User ID=sa;password=examlyMssql@123;
server=localhost;Database=appdb;trusted_connection=false;Persist Security Info=False;Encrypt=False";

Frontend Requirement:



Generating Angular Module, Services, and Components for Crop Management

Create a folder - path (src/app/models)

Models Implementation:

. Crop Model Interface - crop.model.ts - (src/app/models)

· Properties to be implemented:

. CropId (number): Unique identifier for each crop.

. Name (string): Name of the crop.

. Type (string): Type of the crop (e.g., cereal, legume).

· Season (string): The season in which the crop is grown (e.g, summer, winter).

. HarvestTimelnDays (number): The time in days required to harvest the crop after planting.

. Yieldinkg (number): The expected yield of the crop in kilograms.

Services Implementation:

. Create a folder - path (src/app/services)

. CropService - crop.service.ts - (src/app/services) - Command: npx ng g s services/crop

. Declare a public apiUri property and set it to the API URL of the Crop API:

. public apiUri = https://8080 -..... premiumproject.examly.io

. Use HttpClient to interact with the backend API.

. Create the following methods to interact with the backend API in order to perform operations on the Crop entity.

. addCrop(crop: Crop): Observable<Crop> - POST - Gets the crop data from the Crop form and calls the REST API with the endpoint apiUrl/api/Crop.

. getCrops(): Observable<Crop[]> - GET - Returns all crops from the REST API with the endpoint apiUrl/api/Crop.

Component Implementation:

HeaderComponent - (src/app/header) - Command: npx ng g component header

Create a header companent that displays the "Crop Management App" title within the <hl> tag and a navigation menu with the following links:

. Add New Crop - when clicked, navigates to the /addNewCrop route (use the routerLink directive)

. View Crops - when clicked, navigates to the /viewCrops route (use the routerlink directive)

CropFormComponent - (src/app/crop-form) - Command: npx ng g component crop-form

crop-form.component.html

FrontEnd URL = ["https://8081 -**********. exomly.jo/addNewCrop"]

Create a crop-form component with the following requirements:

The component should have a form to create a new crop using a Template-driven form.

The form should have the following fields:

. Crop Nome - name = name, id = cropName

. Crop Type - name = type, id = cropType

.Planting Season - name = season, id = plantingSeason

Harvest Time (days) - name = harvestTimelnDays, id = harvestTime

. Yield (kg) - nome = yieldinkg, id = yield

All fieids are required. The form should display error messages when fields are empty, and the form is submitted. Each input field is wrapped in a tag. Inside each div, there should be another with the class error-message to display validation errors.

Validation errors should be displayed using the *ngif directive, and errors should only appear if the form has been submitted and the respective field is invalid.

. Crop Name = [ErrorMessage = Crop Name is required]

. Crop Type= [ErrorMessage = Crop Type is required]

. Planting Season = [ErrorMessage = Planting Season isrequired]

Harvest Time = [ErrorMessage = Harvest Time is required]

. Yield = [ErrorMessage = Yield is required]

The form should have a button "Add Crop" to submit the form. The button must be inside the form with type='submit". It should call the addCrop() method from the crop- form.component.ts.

NOTE: All required error messages should display when the Add Crop button is clicked directly. Additionally, the "Add Crop

If valid, add the crop using the addCrop method from the CropService. . On success, reset the for method and display an error failure to add the crop.

Implement the isFieldinvalid which checks if a specific form display an error message. . Define the isValidCrop() meth required fields of the crop for .

Implement the resetForm() m are fun to use. fields to their initial values. . Implement the reset Messages success and error messages submission.

CroplistComponent (src/app/crop-list) - Command: npx ng g component crop-list

FrontEnd URL = ['https://8081- I examly.io/viewCrops"]

crop-list.component.html

Design a component to display all the crops in a table Name, Type, Season, Harvest

This much information only got.

Test cases:

1.)Frontend AddCrop_form_exists_and_input_fields_present_in_Add_crop

2.)Frontend Crop_table_header content

3.)Frontend Verify_required_validation_on_Add_Crop_button

4.)Frontend should_create_crop_form_component

5.) Frontend crop_form_component_should_call_add_crop_method _on_post

6.)Frontend crop form_component_should_define_addCrop_method

7.) Frontend should_create_crop_list_component

8.)Frontend crop list component should_call_loadCrops_on_ngonl nit

9.)Frontend should_create crop_list_component

10.)Frontend_crop_list_component_should_call_loadCrops_on ngoni nit

11.)Frontend_crop_list_component_should_define_loadCrops_method

12.)Frontend should create crop instance

13.)Frontend should_create_CropService

14.)Frontend CropService should_add_a_crop_and_return_it

15.)Frontend CropService_should_get_all_crops

16.)Frontend should create header component

17.)Backend Test_Crop_Properties Should_Exist

18.)Backend Test Get_All_Crops_Returns HttpStatusCode OK

19.)Backend Test Posts Crop Returns HttpStatusCode_OK

Additionally, the "Add Crop" button should only add the crop if the form is valid.
If valid:
- Call the addCrop() method from the CropService to submit the crop data to the backend.
- On successful response:
  - Reset the form using resetForm().
  - Display a success message: "Crop added successfully!"
- On error:
  - Display an error message: "Failed to add crop. Please try again."

Implement the following helper methods inside crop-form.component.ts:

1. isFieldInvalid(field: NgModel): boolean
   - Checks if a specific form field is invalid and has been touched or submitted.
   - Used to conditionally display validation error messages using *ngIf.

2. isValidCrop(): boolean
   - Checks whether all required crop fields are valid.
   - Used before submitting to ensure form validity.

3. resetForm(form: NgForm): void
   - Resets the form fields to their initial state.
   - Clears any success or error messages.

Implement logic to:
- Display success message in a with class success-message
- Display error message in a with class error-message
- Messages should appear only after form submission attempt