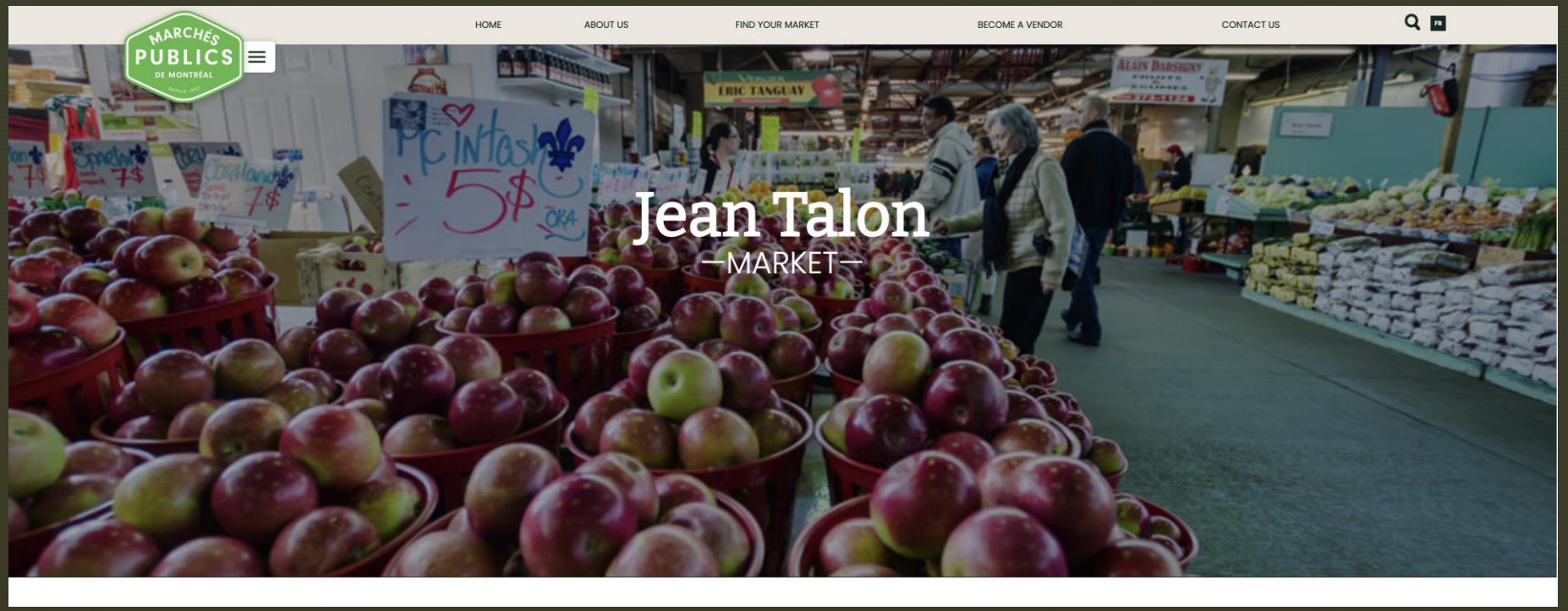
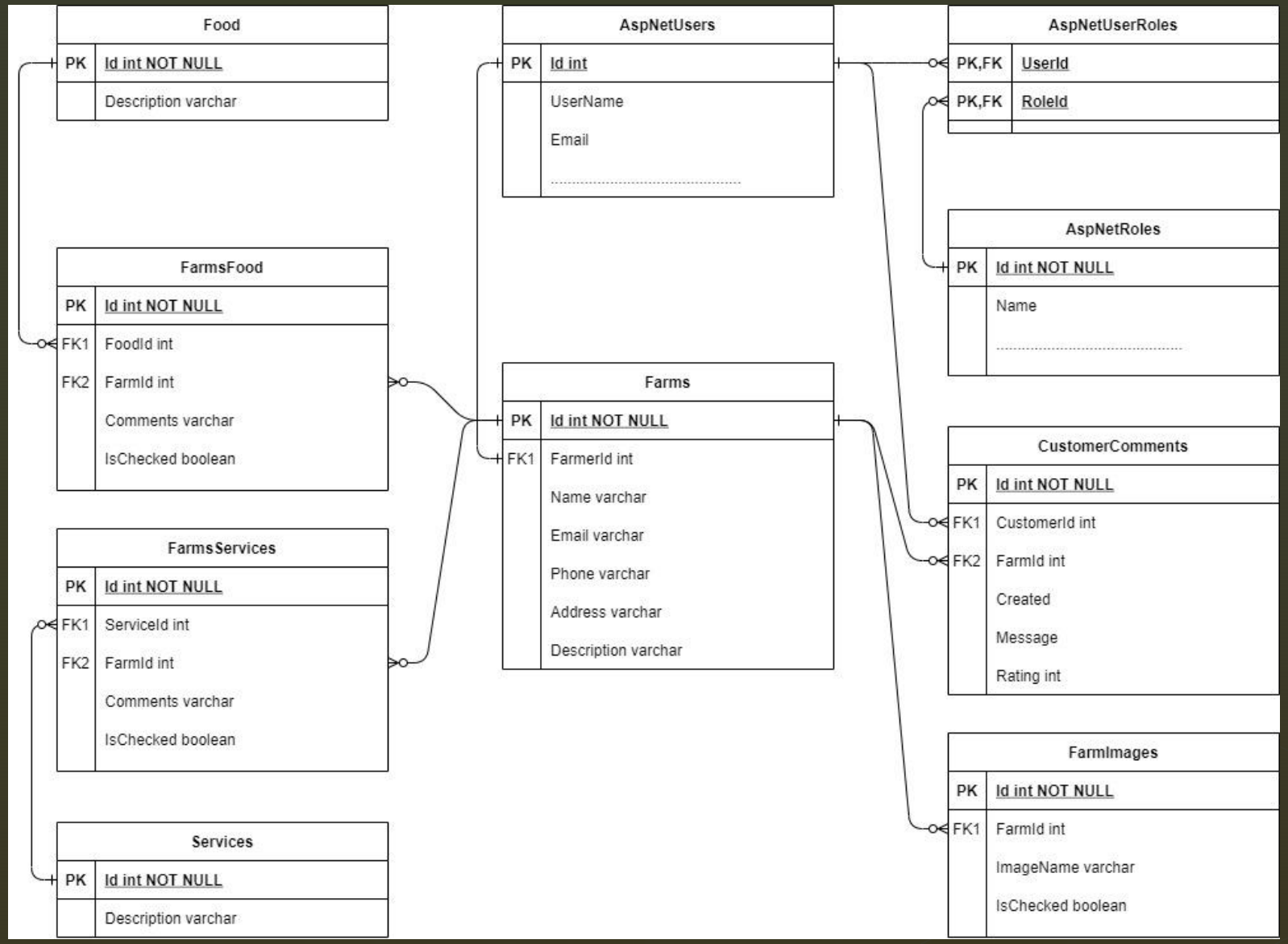
That's My Farmer

- Team members:
- Emma De Barros
- Andrey Lyamkin
- Anastassia Titova

Prototype...


MARCHÉS PUBLICS DE MONTRÉAL
HOME
ABOUT US
FIND YOUR MARKET
BECOME A VENDOR
CONTACT US
FR
Jean Talon
MARKET

DB Structure



Overview of implemented functionnalities

- ✓ Layout/Index
- ✓ Search/Search Result
- ✓ User Profile + CRUD
- ✓ Farmer Profile + CRUD
- ✓ Add Posts on Farm Details View
- ✓ Manage Posts depending on User Role + CRUD

Major Challenges Encountered

- Querying related data (FK)
- Global search bar
- CRUD on IdentityUser Class

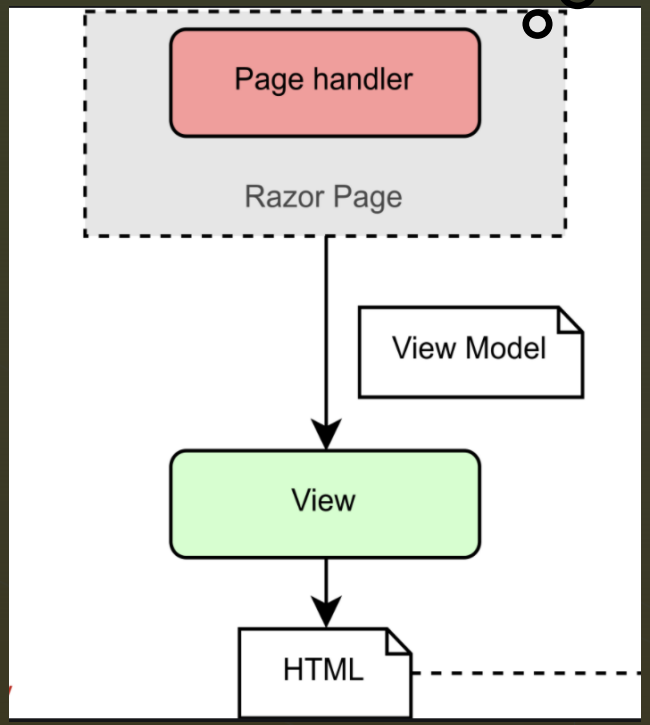
"Auto-magically happens", as a wise man once said...

```
user.UserName = Input.Email;
user.Email = Input.Email;

// sign user out and update
await signInManager.SignOutAsync();
var result = await userManager.UpdateAsync(user);
```

```
user.UserName = Input.Username;
```

```
// Create new object from IdentityUser
var user = new IdentityUser {UserName = Input.Email, Email = Input.Email, EmailConfirmed = true};
// Create new user in the AspNetUsers table
var createUserResult = await userManager.CreateAsync(user, Input.Password);
```

Implementing a Global Search Box... 🤔



Solution...

```
<button asp-page="/SearchResult"
```

Step 1

```
<form class="form-inline ml-lg-3 mr-auto" method="GET">
    <input class="form-control form-control-sm mr-1" type="search" placeholder="Lookup a farm..."
        aria-label="Search" name="Keywords" asp-route-keywords="Keywords">
    @* value="@ViewData["Keywords"]" *@
    <button asp-page="/SearchResult"
        class="btn btn-sm btn-outline-success my-1 my-sm-0 nav-search z-depth-0" type="submit"><i
            class="fas fa-search"></i></button>
</form>      You, a week ago • Update Index
```

Step 2

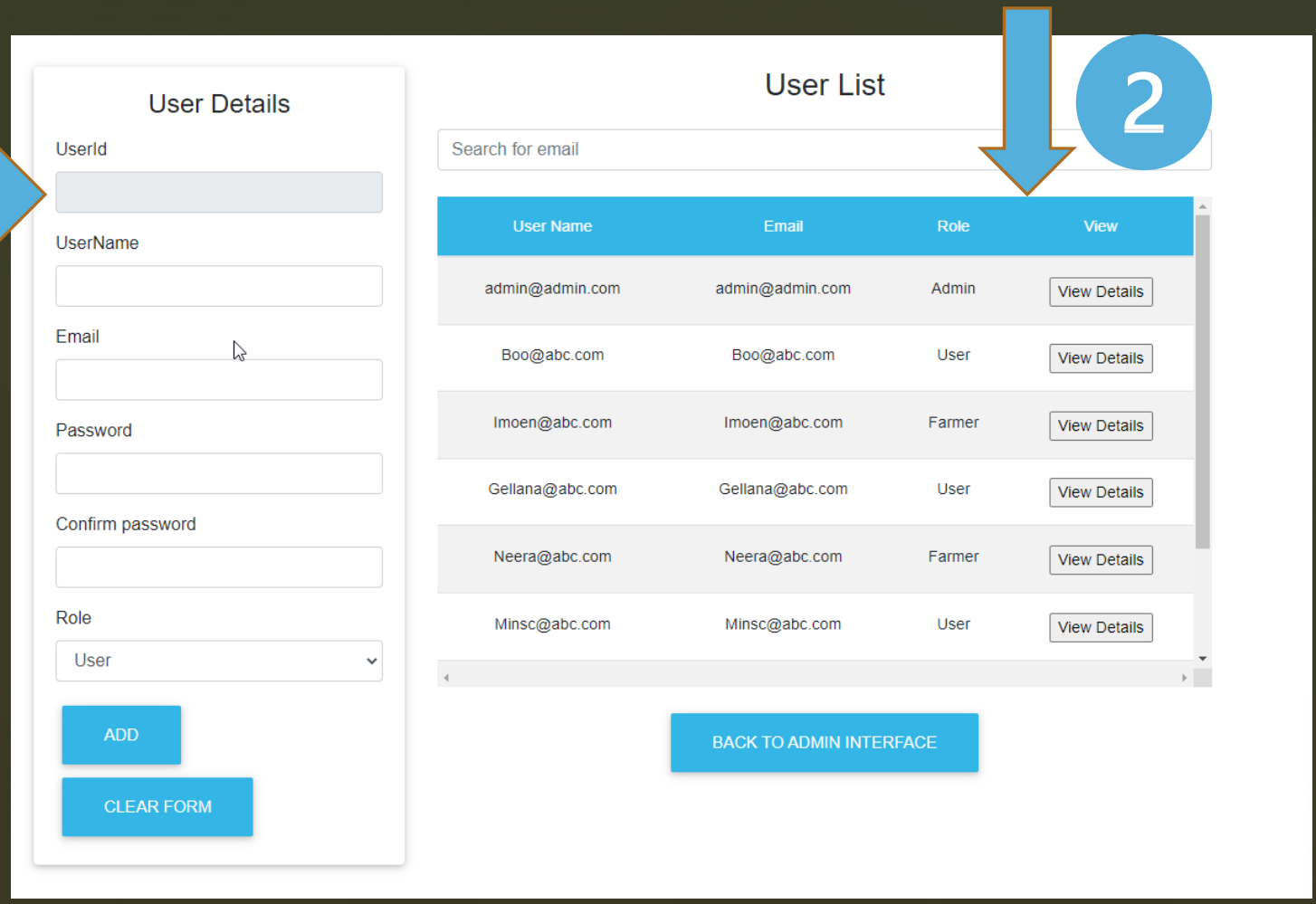
```
    [BindProperty(SupportsGet = true)]
    4 references
    public string keywords { get; set; }
```

Razor Pages is a server-side, page-focused framework

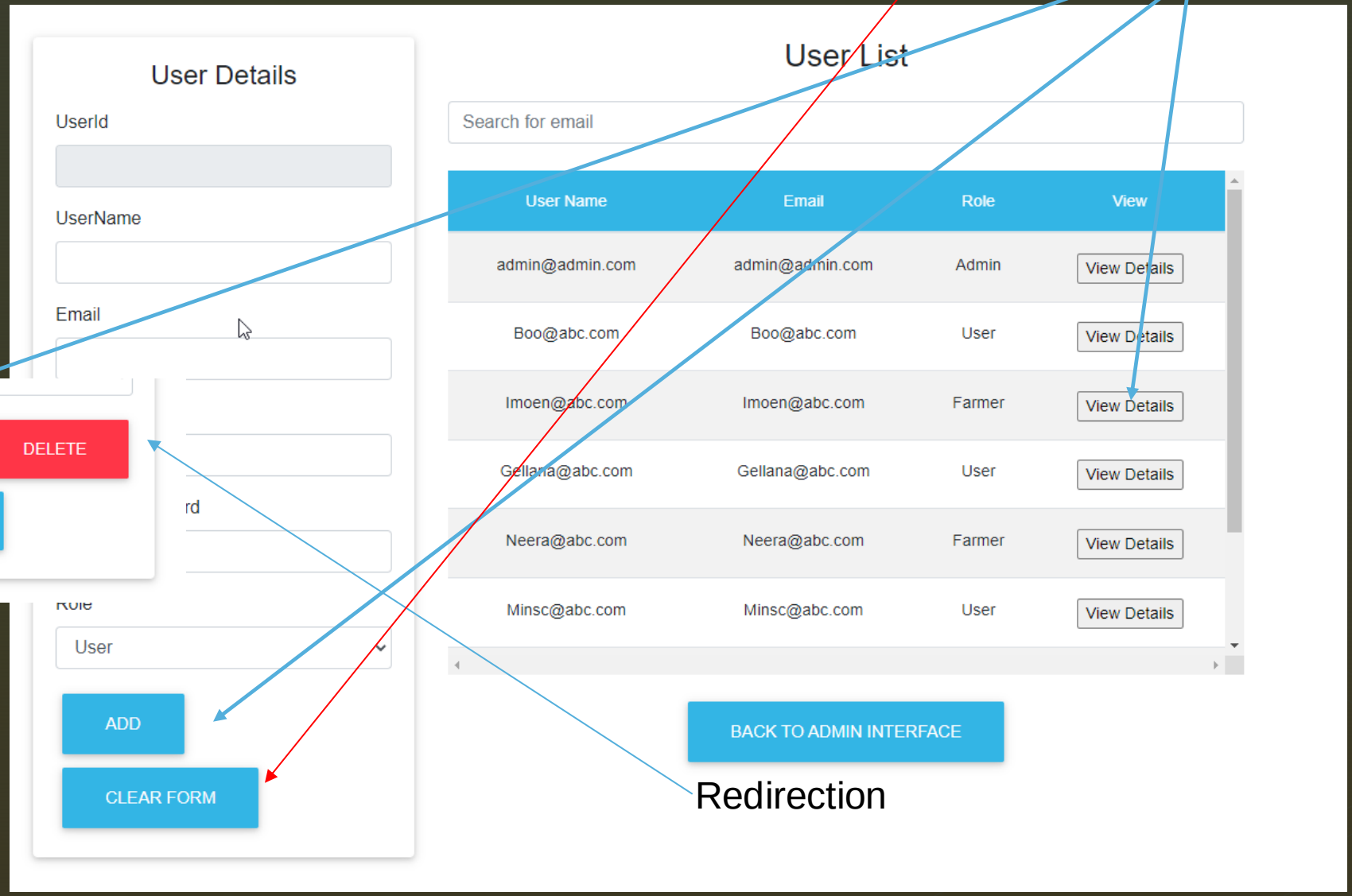
6 table
6 views
6 adds
6 updates
6 deletes
******
30 pages * 2 =
60 files

Can the amount of pages be reduced?

2 forms per page



GET or POST ?


User Details
UserId
UserName
Email
UPDATE
DELETE
CLEAR FORM
User
ADD
CLEAR FORM
User List
Search for email
User Name
Email
Role
View
admin@admin.com
admin@admin.com
Admin
View Details
Boo@abc.com
Boo@abc.com
User
View Details
Imoen@abc.com
Imoen@abc.com
Farmer
View Details
Gellana@abc.com
Gellana@abc.com
User
View Details
Neera@abc.com
Neera@abc.com
Farmer
View Details
Minsc@abc.com
Minsc@abc.com
User
View Details
BACK TO ADMIN INTERFACE
Redirection

GET Request :

Clear Button

```
<form method="GET">
    <div class="btn-group">
        <button type="submit" class="btn btn-info btn-lg mr-4" asp-page-handler="OnGet">Clear
            Form</button>
    </div>
```

```
@if (String.IsNullOrEmpty(Model.Input.UserId))
{
    <button type="submit" class="btn btn-info btn-lg mr-4" asp-page-handler="AddUser">Add</
    button>
}
else
{
    <div class="btn-group">
        <button type="submit" class="btn btn-info btn-lg mr-4" asp-page-handler="UpdateUser"
        asp-route-id="@Model.Input.UserId">Update</button>
        <button type="submit" class="btn btn-danger btn-lg" asp-page="/Admin/DeleteUser"
        asp-route-id="@Model.Input.UserId">Delete</button>
    </div>

}
```

```
    <form method="POST" asp-page-handler="OnPost">
        <td><input type="submit" asp-route-id="@user.Id" value="View Details"></td>
    </form>
/tr>
```

POST Request:

Add Button

Update Button

View Details Button

How are the buttons recognized by Page Controller?

ASP-PAGE-HANDLER

ADD USER

Special Features:

OnPostAddUser

RedirectToPage("/Admin/Users")

```
0 references
public async Task<IActionResult> OnPostAddUser()
{
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email, EmailConfirmed = true };
        var result = await userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            var result2 = await userManager.AddToRoleAsync(user, Input.Role);
            if (result2.Succeeded)
            {
                logger.LogInformation($"User {Input.Email} create a new account with password");
                Message = $"User {Input.Email} new account created";
                return RedirectToPage("/Admin/Users");
            }
            else
            {
                // FIXED: delete the user since role assignment failed, log the event, show error to the user
                await userManager.DeleteAsync(user);
                logger.LogInformation($"User  {Input.Email} new account not created");
                Message = $"User  {Input.Email} new account not created";
                return RedirectToPage("/Admin/Users");
            }
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
        UserList = userManager.Users.ToList();
        return Page();
    }
    else
    {
        UserList = userManager.Users.ToList();
        return Page();
    }
}
```

UPDATE USER

Special features:

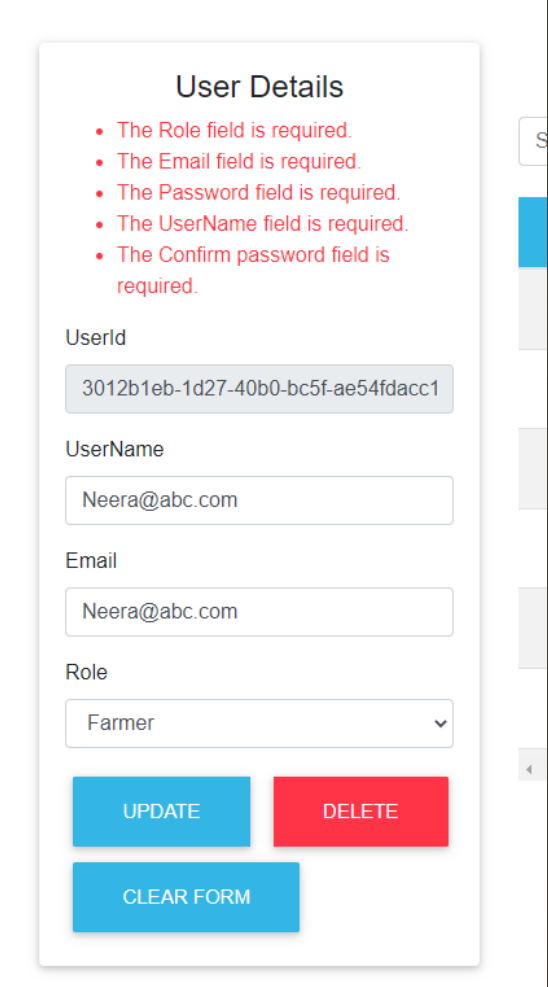
OnPostUpdateUser

Has a parameter
(string id)

Redirection to the same page

```
0 references
public async Task<IActionResult> OnPostUpdateUser(string id)
{
    if (ModelState.IsValid)
    {
        var user = userManager.Users.FirstOrDefault(i => i.Id == id);
        if (user != null)
        {
            user.UserName = Input.Email;
            user.Email = Input.Email;
            var result = await userManager.UpdateAsync(user);
            if (result.Succeeded)
            {
                var oldRole = userManager.GetRolesAsync(user).Result.FirstOrDefault();
                if (oldRole != null)
                {
                    await userManager.RemoveFromRoleAsync(user, oldRole);
                }
                var result2 = await userManager.AddToRoleAsync(user, Input.Role);
                if (result2.Succeeded)
                {
                    logger.LogInformation($"User {Input.Email} updated");
                    Message = $"User {Input.Email} account is updated";
                    return RedirectToPage("/Admin/Users");
                }
            }
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
            Input.UserId = id;
            UserList = userManager.Users.ToList();
            return Page();
        }
        return RedirectToPage("/Admin/Users");
    }
    Input.UserId = id;
    UserList = userManager.Users.ToList();
    return Page();
}
```

VIEW DETAILS - the most challenging part of code on User's Page




```
0 references
public IActionResult OnPost(string id)
{
    var user = userManager.Users.FirstOrDefault(i => i.Id == id);
    Input = new InputModel();
    Input.UserId = user.Id;
    Input.UserName = user.UserName;
    Input.Email = user.Email;
    Input.Password = "FillInBlank123!";
    Input.ConfirmPassword = "FillInBlank123!";
    Input.Role = userManager.GetRolesAsync(user).Result.FirstOrDefault();
    UserList = userManager.Users.ToList();
    RolesList = roleManager.Roles.ToList();

    ModelState.Clear();

    return Page();

}
```

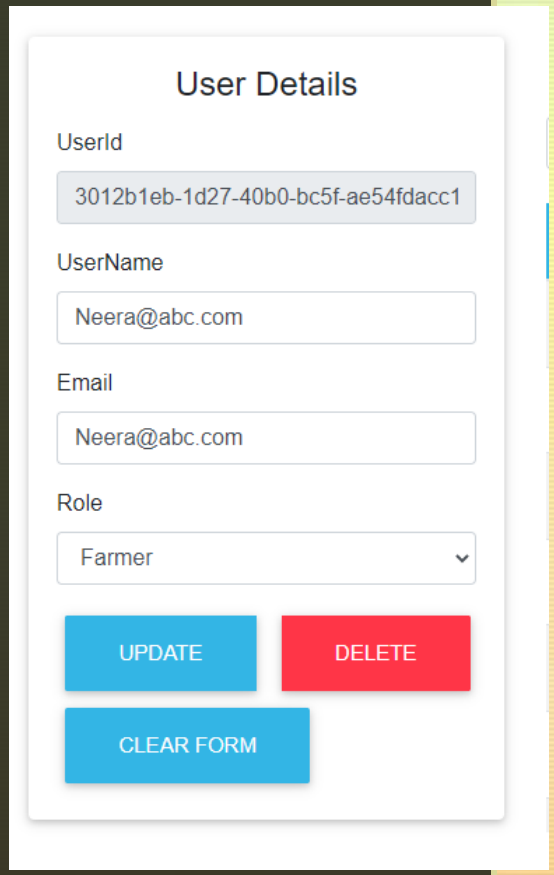
Validation of the form : ModelState Errors Model.Input happen after

Result:

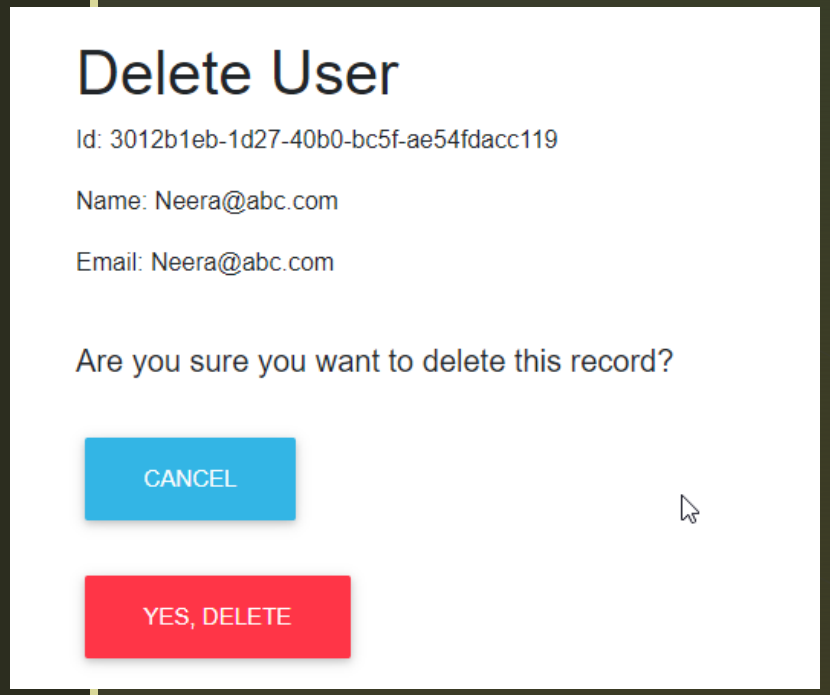
No Model.Input errors but the list of ModelState errors.

Solution:

ModelState.Clear();



DELETE USER




```
0 references
public async Task<IActionResult> OnPostDeleteUser()
{
    var userToDelete = userManager.Users.FirstOrDefault(u => u.Id == Id);
    if (userToDelete == null)
    {
        TempData["message"] = $"User account was not found.";
        return RedirectToPage("/Admin/Users");
    }

    var roleToDelete = userManager.GetRolesAsync(userToDelete).Result.FirstOrDefault();
    if (roleToDelete == "Farmer")
    {
        var farms = db.Farms.Where(f => f.Farmer.Id == Id).ToList();
        if (farms != null)
        {
            foreach (var f in farms)
            {
                var farmFoods = db.FarmsFoods.Where(ff => ff.Farm.Id == f.Id).ToList();
                if (farmFoods != null)
                {
                    db.FarmsFoods.RemoveRange(farmFoods);
                }
                var farmServices = db.FarmsServices.Where(fs => fs.Farm.Id == f.Id).ToList();
                if (farmServices != null)
                {
                    db.FarmsServices.RemoveRange(farmServices);
                }
                db.Farms.Remove(f);
            }
        }
    }
    db.SaveChanges();
    var result1 = await userManager.RemoveFromRoleAsync(userToDelete, roleToDelete);
    if (result1.Succeeded)
    {
        var result = userManager.DeleteAsync(userToDelete);
        if (result.IsCompletedSuccessfully)
        {
            TempData["message"] = $"User {userToDelete.Email} account is deleted.";
            return RedirectToPage("/Admin/Users");
        }
    }
    TempData["message"] = $"User {userToDelete.Email} account was not deleted.";
    return RedirectToPage("/Admin/Users");
}
```

Conclusion:

How many files?

12 pages * 2 = 24 files for 6 tables

Is it the right way to do it?

I'm not sure, but I'm glad I did it this way

What have I learned?

A lot.
How GET and POST request work.
What is the ModelState.
The diffrence between ModelState and Model.
Page handlers for multiple forms or for different CRUD operations on the same form.

Andrey's Task list

- Registration new users
- Login
- Logout
- Add new farm
- Update/Delete farms
- View farm details
- View farms list
- DB ERD

Registration new users

Challenge : 3 groups of users. Each group should be separated from each other. Different functionality and access rules.

- Users – visitors. View farms, manage user profile, add comments.
- Farmer – businessmen. Register new farm, CRUD farms, view other farms.
- Admin – local boss. Manage profiles. Be like a DREDD.

Solution: Microsoft.AspNetCore.Identity.

1. Seed the roles into ASPNetRoles table.
2. Create new IdentityUser into AspNetUsers table.
3. Based on condition Assign role for new user. AspNetUserRoles
4. Based on the role separate users using [Authorize(Roles = "Farmer")] or [Authorize(Roles = "User")]

Seed the roles

Add function in the Startup.cs

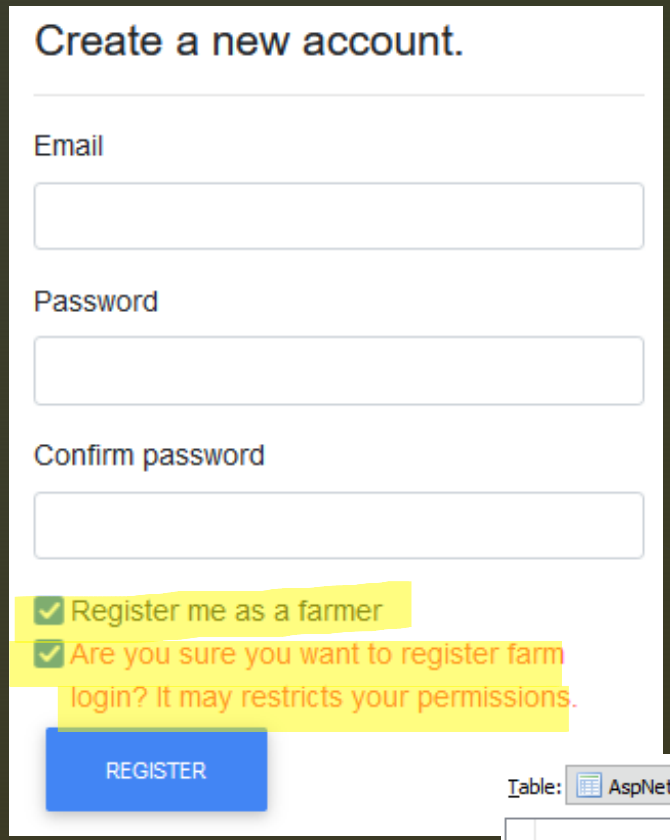
```
1 reference
private void SeedUsersAndRoles(UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager) {
    // create array with roles
    string[] roleNamesList = new string[] { "User", "Admin","Farmer" };

    foreach (string roleName in roleNamesList)
    {
        // check if role not exist
        if (!roleManager.RoleExistsAsync(roleName).Result) {
            // create role object
            IdentityRole role = new IdentityRole();
            // add new role
            role.Name = roleName;
            // save new role in the db
            IdentityResult result = roleManager.CreateAsync(role).Result;
        }
    }
```

Table: AspNetRoles

Filter in any column

	Id	Name	NormalizedName	ConcurrencyStamp
	Filter	Filter	Filter	Filter
1	5ccd7195-1102-4638-a4c9-170c5a4e1efe	User	USER	d7868d96-866f-43e3-8847-fd59bb62003d
2	cc2bb58a-efda-4370-adca-f2a07ca6f48b	Admin	ADMIN	987f3591-a356-419a-a149-c37edbabf398
3	fd8c49ad-44b8-499d-92cd-65e7a7440509	Farmer	FARMER	d49cf062-def0-48a5-9d9b-16c9565994a2

Register users and assign roles




```
// Create new object from IdentityUser
var user = new IdentityUser {UserName = Input.Email, Email = Input.Email, EmailConfirmed = true};
// Create new user in the AspNetUsers table
var createUserResult = await userManager.CreateAsync(user, Input.Password);
if(createUserResult.Succeeded)
{
    // Try to add "User" or Farmer Role to the new user
    IdentityResult addRoleResult = new IdentityResult();
    if(Input.RegisterMeAsFarmer)
    {
        addRoleResult = await userManager.AddToRoleAsync(user, "Farmer");
    } else
    {
        addRoleResult = await userManager.AddToRoleAsync(user, "User");
    }
```

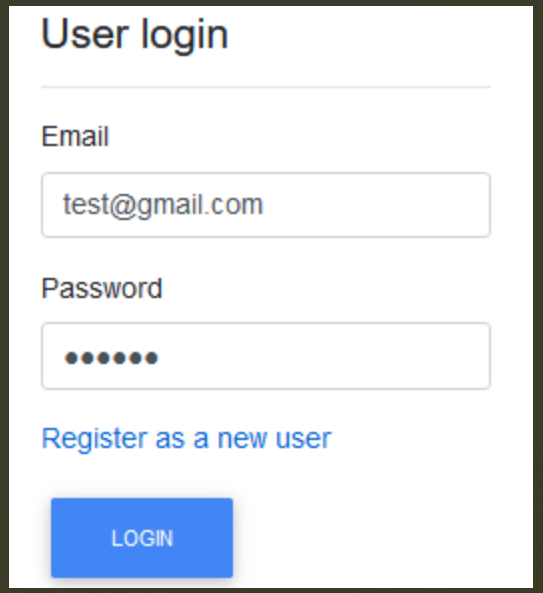
Table: AspNetUsers — Filter in any column

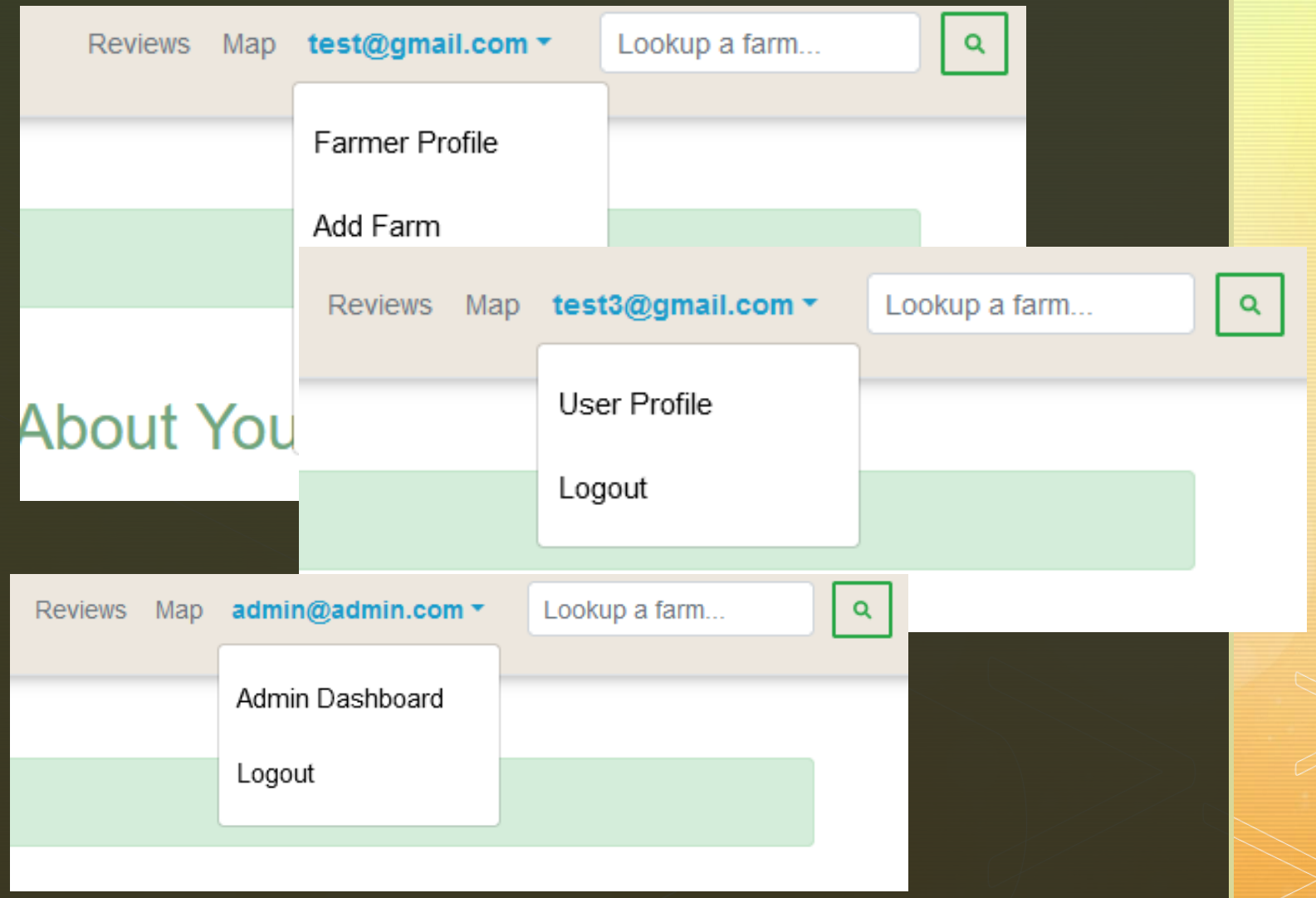
	Id	UserName	NormalizedUserName	Email
	Filter	Filter	Filter	Filter
1	f5525dbe-82ce-4904-8c10-a93df7f46a98	admin@admin.com	ADMIN@ADMIN.COM	admin@admin.com
2	588d15eb-7e63-4aea-b67b-ed536fc5f804	test@gmail.com	TEST@GMAIL.COM	test@gmail.com
		gmail.com	TEST3@GMAIL.COM	test3@gmail.com
		gmail.com	TEST2@GMAIL.COM	test2@gmail.com

Table: AspNetUserRoles

	UserId	RoleId
	Filter	Filter
1	f5525dbe-82ce-4904-8c10-a93df7f46a98	cc2bb58a-efda-4370-adca-f2a07ca6f48b
2	588d15eb-7e63-4aea-b67b-ed536fc5f804	fd8c49ad-44b8-499d-92cd-65e7a7440509
3	4df2390d-b098-4e1b-92e1-9c08d8a449b0	5ccd7195-1102-4638-a4c9-170c5a4e1efe
4	4d5aefb0-b637-41ca-bdd7-fdcb38072b6f	fd8c49ad-44b8-499d-92cd-65e7a7440509

Login





Based on the Role (User, Admin, Farmer) the website displays different options

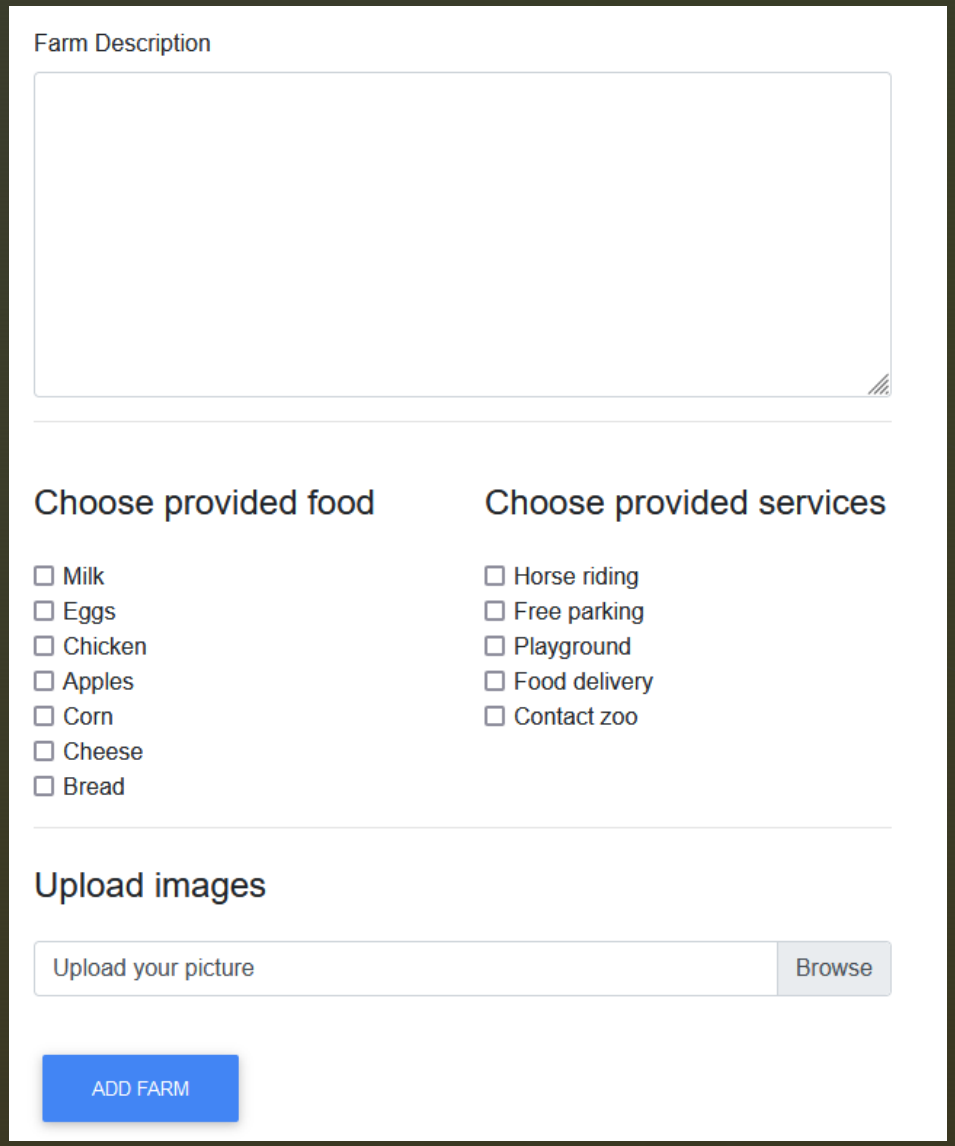
Login. Code behind

```
@if (signInManager.IsSignedIn(User))
{
    @if (Role == "User")
    {
        <li class="nav-item dropdown">
            <a href="#!" class="nav-link text-info font-weight-bold text-muted dropdown-toggle"
        id="navbarDropdownMenuLink" data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false">
                    @User.Identity.Name</a>
            <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
                <a class="dropdown-item" asp-area="" asp-page="/Account/UserProfile">User Profile</
                a>
                <a class="dropdown-item" asp-area="" asp-page="/Account/Logout">Logout</a>

            </div>
        </li>
    }
    else if (Role == "Farmer")
```

```
Role = role;
```

```
@using Mi
@inject S
@inject U
@{
    strir
    if (s
    {
        v
        v
        v
        /
        f
        {
        }
    }
}
```

Add new farm



- **Challenge 1** : Add predefined list of food options and save the state
- **Challenge 2** : Add predefined list of services and save the state
- **Challenge 3** : Add multiple images for farm

Add new farm. Food options. OnGet

Table: Foods

	Id	Description
	Filter	Filter
1	1	Milk
2	2	Eggs
3	3	Chicken
4	4	Apples

```
FoodList = new List<Food>();
FoodList = db.Foods.ToList();
FoodList.ForEach(food => {
    InputModelFarmFood InputsFarmFood = new InputModelFarmFood();
    InputsFarmFood.Id = food.Id;
    InputsFarmFood.Description = food.Description;
    InputsFarmFood.IsChecked = false;
    InputsFarmFoodList.Add(InputsFarmFood);
});
```

```
[BindProperty]
5 references
public List<InputModelFarmFood> InputsFarmFoodList { get; set; }
6 references
public class InputModelFarmFood
{
    [Required]
    4 references
    public int Id { get; set; }
    [Required, MinLength(1), MaxLength(255)]
    4 references
    public string Description { get; set; }
    4 references
    public bool IsChecked { get; set; }
}
```

```
@for (var i = 0; i < @Model.InputsFarmFoodList.Count(); i++)
{
    <div class="form-check">
        <input type="hidden" asp-for="InputsFarmFoodList[i].Id" />
        <input type="hidden" asp-for="InputsFarmFoodList[i].Description" />
        <input asp-for="InputsFarmFoodList[i].IsChecked" class="form-check-input" />
        <label asp-for="InputsFarmFoodList[i].IsChecked" class="form-check-label">
            @Model.InputsFarmFoodList[i].Description
        </label>
    </div>
}
```

Add new farm. Food options. OnPost

Table: Foods

	Id	Description
	Filter	Filter
1	1	Milk
2	2	Eggs

Table: FarmsFoods

	Id	FarmFoodId	FarmId	Comments	IsChecked
	Filter	Filter	Filter	Filter	Filter
1	25	1	7	NULL	1
2	26	2	7	NULL	1
3	27	3	7	NULL	0
4	28	4	7	NULL	0
5	29	5	7	NULL	1
6	30	6	7	NULL	0
7	31	7	7	NULL	1

```
if(FarmId != null)
{
    try {
        // Loop over FoodList and save records in the FarmsFood
        InputsFarmFoodList.ForEach(item => {
            Food foodItem = db.Foods.FirstOrDefault(i => i.Id
            == item.Id); // Get food object from db
            db.FarmsFoods.Add(new FarmsFood{FarmFood =
            foodItem, Farm = NewFarm, IsChecked = item.
            IsChecked}); // Create new record in FarmsFoods
            table
            logger.LogInformation($"New food: {item.
            Description} was added to farm {NewFarm.Name}");
        });
```

Add new farm. Add Images

```
private string handleImageSaving()
{
    string uniqueFileName = null;
    if(Upload !=null) {
        string fileExtension = Path.GetExtension(Upload.FileName).ToLower(); // get file extension
        string[] allowedExtensions = {".jpg",".jpeg",".gif",".png"};
        // check if uploaded file is an image type
        if(!allowedExtensions.Contains(fileExtension)) {
            ModelState.AddModelError(string.Empty, "Only image files (.jpg, .jpeg, .gif, .png) are allowed");
            return null;
        }
        // Physical Path to upload folder
        string uploadsFolder = Path.Combine(webHostEnvironment.WebRootPath, "Uploads");
        // Create unique file name
        uniqueFileName = Guid.NewGuid().ToString() + "_" + Upload.FileName;
        // Full path to file
        string destPath = Path.Combine(uploadsFolder, uniqueFileName);
        try {
            using (var fileStream = new FileStream(destPath, FileMode.Create))
            {
                Upload.CopyTo(fileStream);
            }
        } catch(Exception ex ) when (ex is IOException|| ex is SystemException) {
            ModelState.AddModelError(string.Empty, "Internal error saving the uploaded image");
            return null;
        }
    }
    return uniqueFileName;
}
```

Delete farm.

Table: Farms

	Id	FarmerId	Name
	Filter	Filter	Filter
1	7	588d15eb-7e63-4aea-b67b-ed536fc5f804	Piggy pig farm
2	8	588d15eb-7e63-4aea-b67b-ed536fc5f804	Super old farm

Table: FarmsFoods

	Id	FarmFoodId	FarmId	Comments	IsChecked
	Filter	Filter	Filter	Filter	Filter
1	25	1	7	NULL	1
2	26				
3	27				

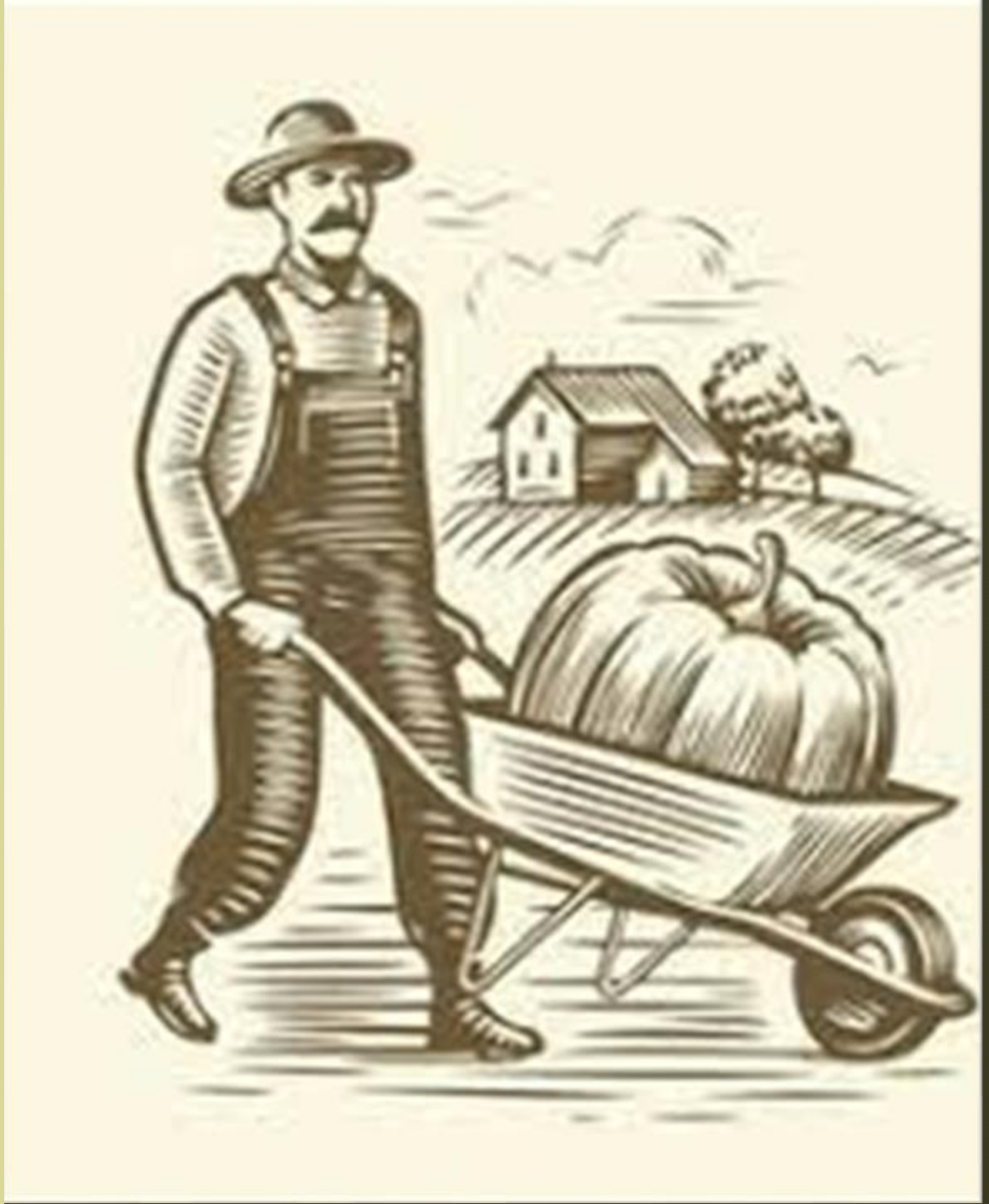
Table: FarmsServices

	Id	FarmServiceId	FarmId	Comments	IsChecked
	Filter	Filter	Filter	Filter	Filter
1	13	1	7	NULL	0
2	14	2	7	NULL	0
3	15	3	7	NULL	1

Table: FarmImages

	Id	FarmId	ImageName	isChecked
	Filter	Filter	Filter	Filter
1	6	7	917e7259-ea66-4df6-bfb7-3c027c37d144_pig.jpg	0
2	7	8	8930c920-457c-4e51-bbfb-4345b914638f_oldcar.jpg	0
3	8	9	ae8495c9-0b69-46f8-b480-f968dd1ec514_sheeps.jpg	0

```
public IActionResult OnPost()
{
    Farm farmToDelete = db.Farms.FirstOrDefault(Farm=>Farm.Id==Id);
    if (farmToDelete == null)
    {
        logger.LogInformation($"Farm with is {Id} was not found.");
        return RedirectToPage("/NotFound");
    }
    //TODO: Step back if one of transactions failed
    // Get all records from FarmsFood and delete them
    IList<FarmsFood> farmsFoodToDelete = db.FarmsFoods.Where(item => item.Farm
    == farmToDelete).ToList();
    db.FarmsFoods.RemoveRange(farmsFoodToDelete);
    // Get all records from FarmsService and delete them
    IList<FarmsService> farmsServicesToDelete = db.FarmsServices.Where(item =>
    item.Farm == farmToDelete).ToList();
    db.FarmsServices.RemoveRange(farmsServicesToDelete);
    // Get all records from FarmImage and delete them
    IList<FarmImage> farmsImagesToDelete = db.FarmImages.Where(item => item.
    Farm == farmToDelete).ToList();
    db.FarmImages.RemoveRange(farmsImagesToDelete);
    // Remove the farm
    db.Farms.Remove(farmToDelete);
    db.SaveChanges();
    TempData["message"] = $"The farm {farmToDelete.Name} was deleted";
    return RedirectToPage("/FarmsList");
}
```

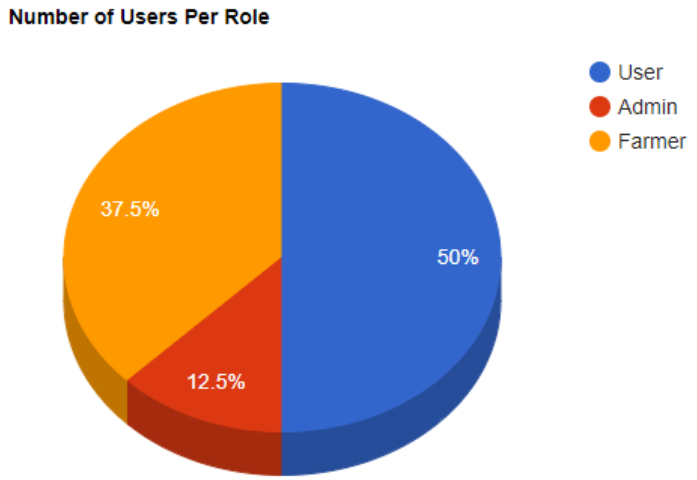


Future Work...

- Farm location – Google Maps
- Rating with stars
- Display top 3 best rated farms on /Index carousel
- Load multiple images for farm

Not included in the presentation...

GOOGLE CHART




```
7 references
public class UserRoleCount
{
    2 references
    public string Role { get; set; }
    2 references
    public int Count { get; set; }
}

3 references
public List<UserRoleCount> UrcList = new List<UserRoleCount>();
```

```
2 references
public List<UserRoleCount> CountUserPerRole()
{
    var RolesList = roleManager.Roles;
    List<UserRoleCount> urcList = new List<UserRoleCount>();
    foreach (var r in RolesList)
    {
        string role = r.Name;
        var usersList = userManager.GetUsersInRoleAsync(r.Name).Result;
        int count = usersList.Count;
        UserRoleCount urc = new UserRoleCount();
        urc.Role = role;
        urc.Count = count;
        urcList.Add(urc);
    }
    return urcList;
}
```

Google chart with Identity User:
- Create new class with fields – Role and Count
- Method CountUserPerRole() - returns a List of Roles with the Count of Users per Role

Main Difference from PHP

```
0 references
public IActionResult OnGetChartData()
{
    UrcList = CountUserPerRole();
   var json = UrcList.ToGoogleDataTable()
        .NewColumn(new Column(ColumnType.String, "Role"), x => x.Role)
        .NewColumn(new Column(ColumnType.Number, "Count"), x => x.Count)
        .Build()
        .GetJson();

    return Content(json);
}
```

To the View we pass JSON (not a List)

Use AJAX to load JSON

DrawTable(jsonData)

Difference:

Add new library:
Google.DataTable.Net.Wrapper

Create table on Page Controler side

```
function drawChart() {

    // Create the data table.
    var jsonData = $.ajax({
        url: "/Admin/Index?handler=ChartData",
        dataType: "json",
        async: false
    }).responseText;

    // Create the data table.
    var data = new google.visualization.DataTable(jsonData);

    // Set chart options
    var options = {
        'title': 'Number of Users Per Role',
        is3D: true,
    };
```