Certainly! Below is an updated explanation, organized according to the file directory structure you shared. It includes specific steps for creating the necessary controllers within the appropriate areas of your project.

**Step 2: Set Up Project Structure**

**1. Create the Project in Visual Studio**

Follow the steps mentioned earlier to create a new **ASP.NET Core MVC** project named CruiseShip.

**Step 3: Create Areas in the Project**

ASP.NET Core MVC supports a modular approach through **Areas**, which help organize different parts of the application into separate functional sections. For this project, you will create the following areas:

**1. Admin Area**

* This area will manage all the administrative functionalities, such as managing facilities, rooms, and bookings.

**2. User Area**

* This area will manage functionalities for users (voyagers), such as registration, login, and making bookings.

**Steps to Create Areas:**

1. In the **Solution Explorer**, right-click on the **Areas** folder (create it if it doesn’t exist).
2. Select **Add > New Folder** and create the following folders:
   * **Admin**: For administrative functionalities.
   * **User**: For user functionalities.

**Step 4: Create Models**

For both the **Admin** and **User** areas, create models to represent the data being handled. These models should go inside the **Models** folder.

**Models to Create:**

1. **Facility Model**: Represents cruise ship facilities (Admin).
2. **Room Model**: Represents rooms available on the cruise ship (Admin).
3. **Booking Model**: Represents user bookings (User).

Example models (same as previous explanation) should be created inside the **Models** folder.

**Step 5: Set Up Database Context**

Create a class to handle the database connection. Place this inside the **Data** folder.

**1. Create ApplicationDbContext**

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Facility> Facilities { get; set; }

public DbSet<Room> Rooms { get; set; }

public DbSet<Booking> Bookings { get; set; }

}

**Step 6: Create Controllers for Admin and User Areas**

**Admin Area**

1. **Create FacilityController**
   * Right-click on the **Admin** folder under **Areas** and select **Add > New Folder**. Name it Controllers.
   * Inside the **Controllers** folder, create a new **MVC Controller - Empty** and name it FacilityController.

Example **FacilityController** for Admin:

public class FacilityController : Controller

{

private readonly ApplicationDbContext \_context;

public FacilityController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Admin/Facility

public async Task<IActionResult> Index()

{

var facilities = await \_context.Facilities.ToListAsync();

return View(facilities);

}

// GET: Admin/Facility/Create

public IActionResult Create()

{

return View();

}

// POST: Admin/Facility/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("FacilityId,Name,Description,ImageUrl")] Facility facility)

{

if (ModelState.IsValid)

{

\_context.Add(facility);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(facility);

}

}

1. **Create RoomController**
   * Similarly, create a RoomController for managing rooms on the cruise ship.
2. **Create BookingController**
   * The BookingController in the **Admin** area can manage administrative functions like viewing all bookings.

**User Area**

1. **Create AccountController** (For User Authentication)
   * In the **User** folder under **Areas**, create a **Controllers** folder.
   * Create a new **MVC Controller - Empty** named AccountController to handle user registration, login, and profile management.

Example **AccountController** for User:

public class AccountController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signInManager;

public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

// GET: User/Account/Register

public IActionResult Register()

{

return View();

}

// POST: User/Account/Register

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new IdentityUser { UserName = model.Email, Email = model.Email };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, isPersistent: false);

return RedirectToAction(nameof(HomeController.Index), "Home");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

return View(model);

}

// GET: User/Account/Login

public IActionResult Login()

{

return View();

}

// POST: User/Account/Login

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

return RedirectToAction(nameof(HomeController.Index), "Home");

}

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

}

return View(model);

}

}

1. **Create BookingController** (For User Bookings)
   * In the **User** folder, create a **BookingController** to manage user bookings.

Example **BookingController** for User:

public class BookingController : Controller

{

private readonly ApplicationDbContext \_context;

public BookingController(ApplicationDbContext context)

{

\_context = context;

}

// GET: User/Booking/MakeBooking

public IActionResult MakeBooking()

{

return View();

}

// POST: User/Booking/MakeBooking

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> MakeBooking(Booking booking)

{

if (ModelState.IsValid)

{

\_context.Add(booking);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(booking);

}

// GET: User/Booking/ViewBookings

public async Task<IActionResult> ViewBookings()

{

var bookings = await \_context.Bookings.ToListAsync();

return View(bookings);

}

}

**Step 7: Set Up Views for Admin and User Areas**

1. **Admin Views**:
   * Inside the **Views/Areas/Admin/Facility** folder, create views like **Index.cshtml**, **Create.cshtml**, and others for facility management.
   * Similarly, create views for managing rooms and bookings.
2. **User Views**:
   * Inside the **Views/Areas/User/Account** folder, create views for user registration, login, and profile management.
   * Inside the **Views/Areas/User/Booking** folder, create views for making and viewing bookings.

**Step 8: Run the Application**

* Press **Ctrl + F5** to build and run the application. Navigate to the appropriate areas for both the admin and user functionalities.

By following these steps, you will organize your project into **Admin** and **User** areas, with controllers handling administrative and user-related functionalities. The project will now be modular, making it easier to manage different sections of the cruise ship management system.