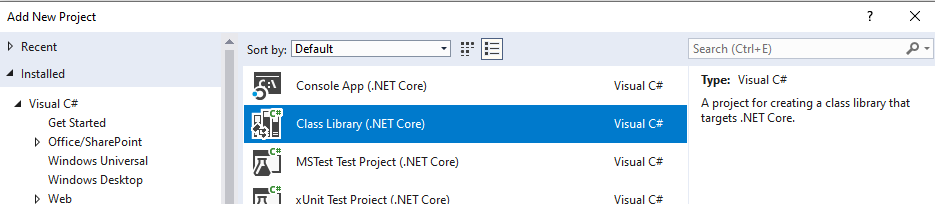
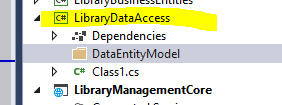
**Add DataAccess Layer as Class Library Project**

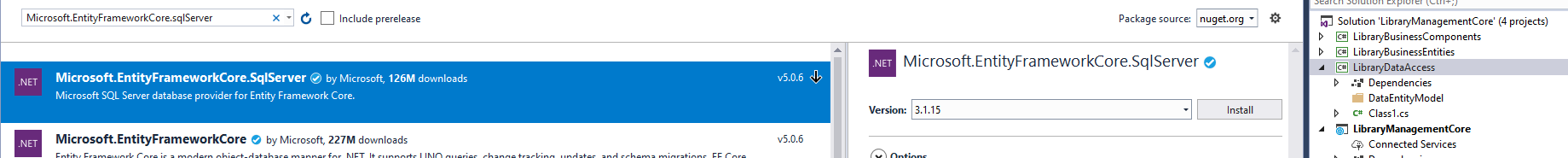




**Install Entity Framework to Class Library Project**

Just install Microsoft.EntityFrameworkCore.sqlServer

**For reference** : <https://www.youtube.com/watch?v=8aHzSx-inDE>



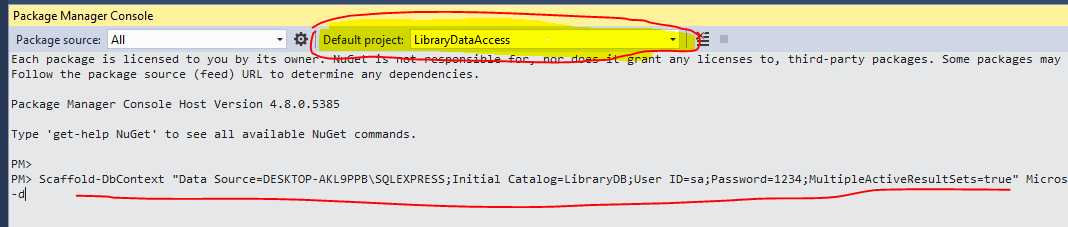
Note: Must Install the same package version of default Website Microsoft.EntityFrameworkCore.sqlServer version.

**For Entity Framework Entity Data Model**

“Class Library (.NET Standard)" doesn’t support ADO.net Entity Data Model.

"Class Library (.NET Framework)" [ which is normal old dot net framework 4.5 ] support ADO.net Entity Data Model.

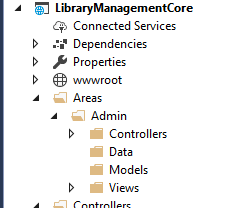
To create data model, open Package Manager Console, Select Default Project as DataAccessProject and then run the following command



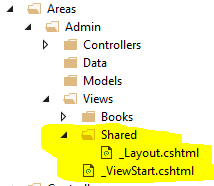
Scaffold-DbContext "Data Source=DESKTOP-AKL9PPB\SQLEXPRESS;Initial Catalog=LibraryDB;User ID=sa;Password=1234;MultipleActiveResultSets=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir DataEntityModel -f –d

**How to Add Areas in ASP.Net Core MVC Project**

Step 1 ) Right click on Project ->> Add Areas 🡪> Enter New Area name (example – Admin ) and press OK.



Step 2 ) Add following two highlighted Views



Step 3 )

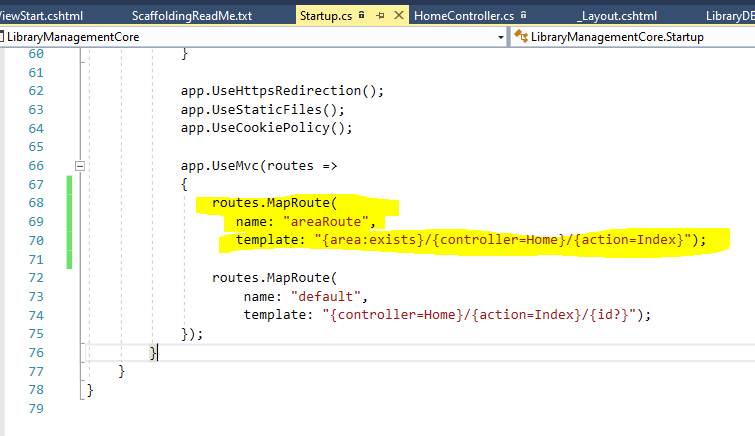
\_ViewStart.cshtml should be like this

@{

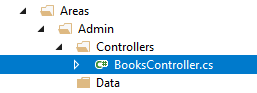
Layout = "~/Areas/Admin/Views/Shared/\_Layout.cshtml";

}

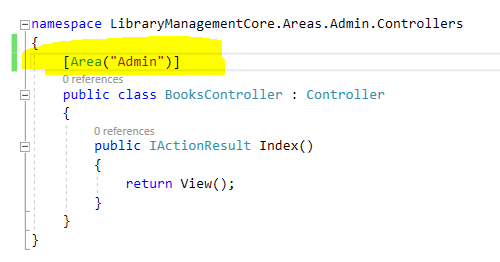
Step 4 ) Inside **Startup.cs** add following Code



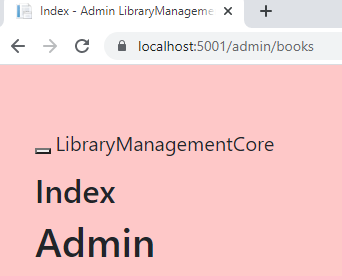
Step 5 ) Add needed Controllers



Step 6 ) Must Add the following Attributes in the Controller



Step 7 ) You can access the page by the url <https://localhost:5001/admin/books>



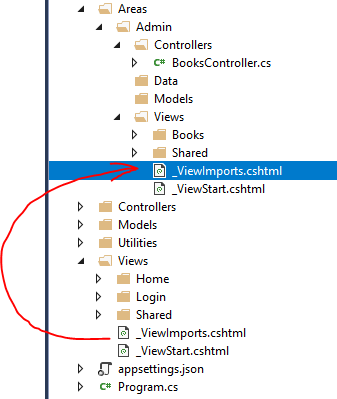
**How to Add Partial View inside any area Areas in ASP.Net Core MVC Project**

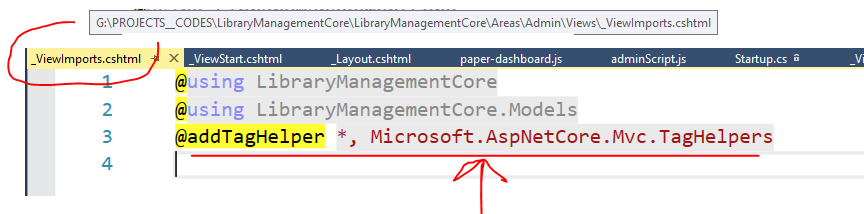
The syntax to use the Partial view using asp.net tag helper

<**partial** **name**="~/Areas/Admin/Views/Shared/\_FooterPartial.cshtml" />

But the thing is, if you only use that syntax, then it will not work.

In order to solve this copy the \_ViewImports.cshtml file from View and add into ~/Areas/<Area\_Name>/Views path.





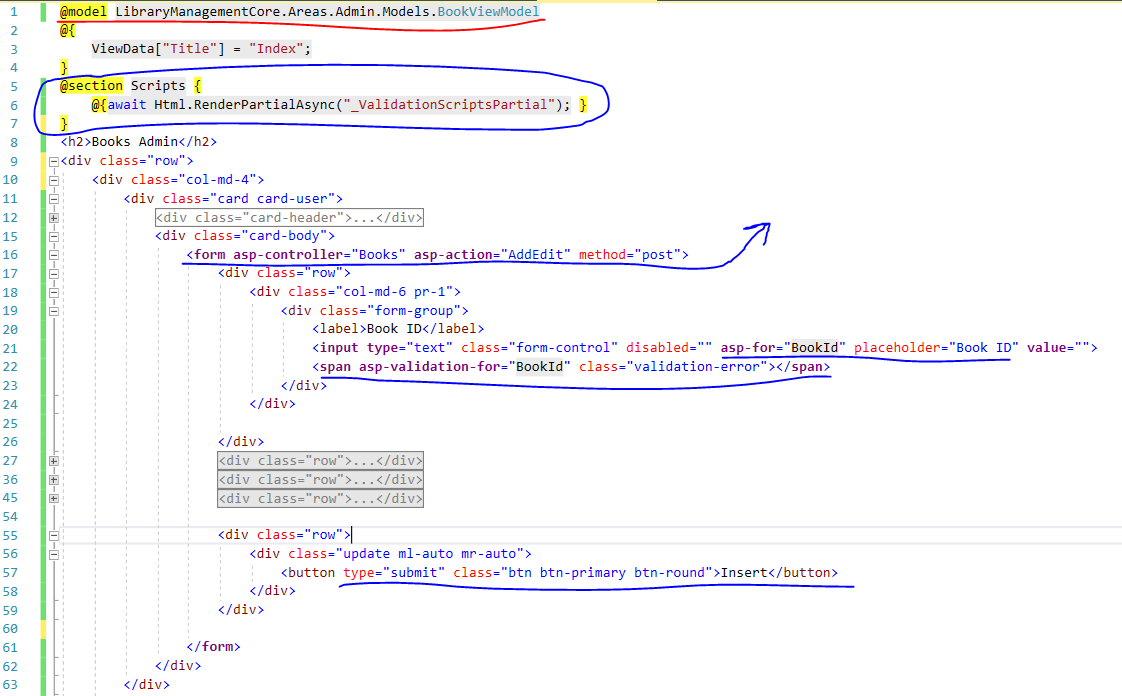
Because -

According to official [docs](https://docs.asp.net/en/latest/mvc/views/tag-helpers/intro.html#addtaghelper-makes-tag-helpers-available):

The @addTagHelper directive makes Tag Helpers available to the view. In this case, the view file is Views/\_ViewImports.cshtml, which by default is inherited by all view files in the Views folder and sub-directories; making Tag Helpers available. The code above uses the wildcard syntax (“\*”) to specify that all Tag Helpers in the specified assembly (Microsoft.AspNetCore.Mvc.TagHelpers) will be available to every view file in the Views directory or sub-directory.

If you use one layout per area, to use built-in tag helpers you should add \_ViewImports.cshtml in ~/Areas/Name/Views/ folder(If you use shared layout you don't need. )

**How to use jQueryUnobtrusive for Model Validation in ASP.Net Core MVC Project**

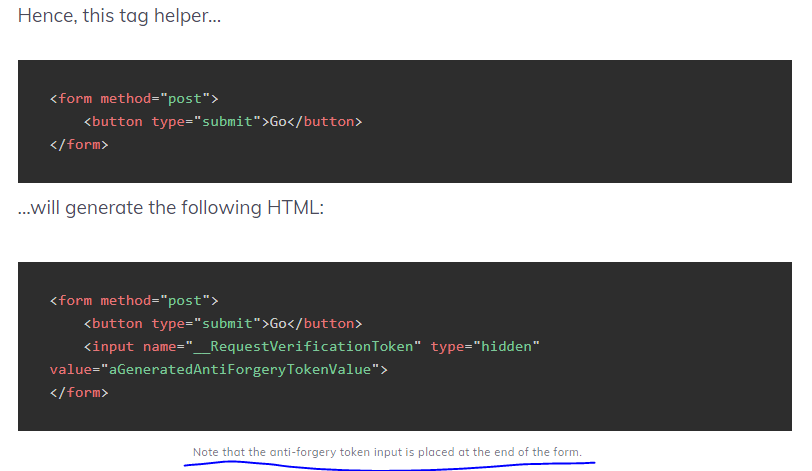


You can help from below :

<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0>

**Anti-Forgery Token in ASP.Net Core MVC Project**

By default, new ASP.NET 5 Razor Pages apps are already equipped with anti-forgery tokens and corresponding validation. On the page, the [form tag helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-3.1#the-form-tag-helper) will automatically render a hidden field containing an anti-forgery token.



And in Post Action



## Opting-Out of Anti-Forgery Token Validation

The first way to opt-out of using anti-forgery token validation is to do so globally by adding a convention to RazorPagesOptions in the Startup.cs file:

public void ConfigureServices(IServiceCollection services)

{

services.AddRazorPages()

.AddRazorPagesOptions(options =>

{

options.Conventions

.ConfigureFilter(new IgnoreAntiforgeryTokenAttribute());

});

}

This inserts an [IgnoreAntiforgeryTokenAttribute](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.ignoreantiforgerytokenattribute?view=aspnetcore-3.1) into our request pipeline, which skips the validation of the token for every request (note that the token will still be generated in our forms, just not validated on the server side).

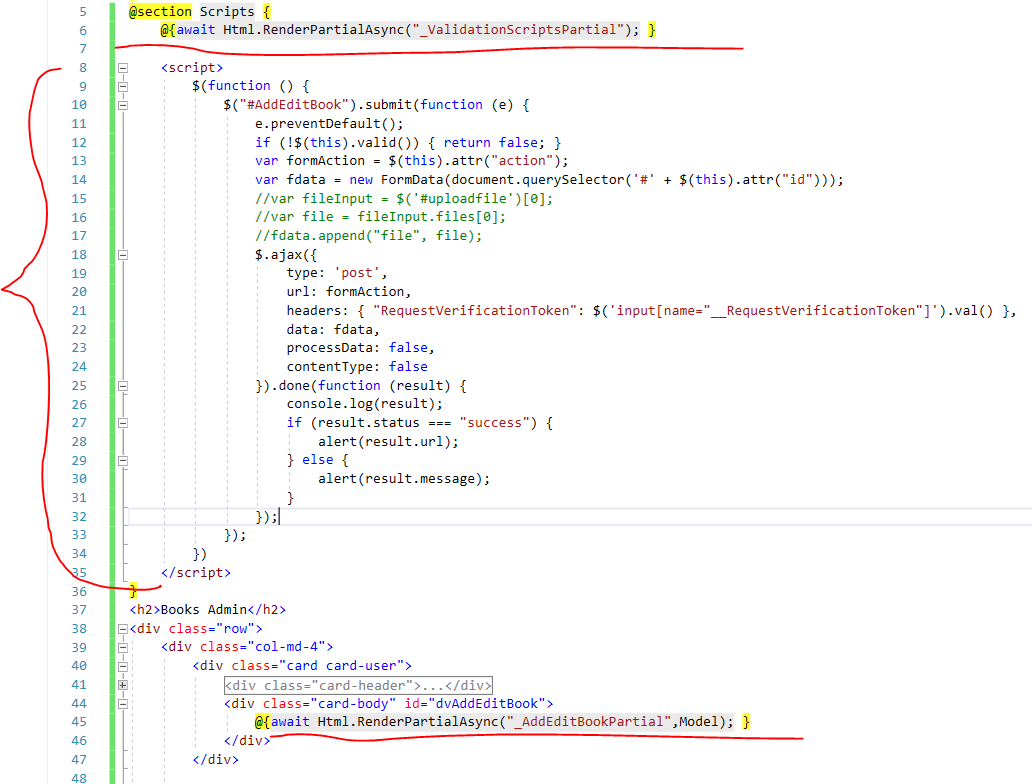
**Note url :** <https://exceptionnotfound.net/using-anti-forgery-tokens-in-asp-net-core-razor-pages/>

If you would also like the form to not render the hidden anti-forgery token field, you can disable that in the form tag helper:

<form method="post" asp-antiforgery="false"></form>

In this way, we can have fine-grained control over which pages use anti-forgery validation, and which do not.

**Anti-Forgery Token for ajax API call in ASP.Net Core MVC Project**



**Usage of Bind Attribute and Async await in ASP.Net Core MVC Project**

**IBookRepository.cs in LibraryDataAccess Layer**

public interface IBookRepository

{

Task<Book> GetBookById(string BookId);

Task<IEnumerable<Book>> GetBooks();

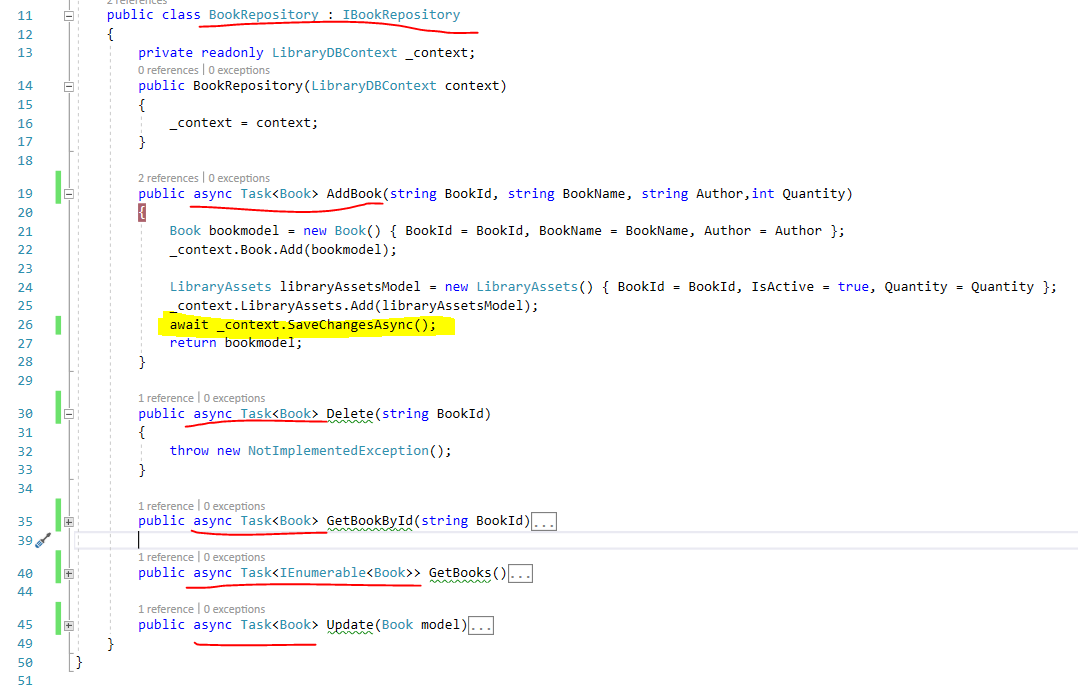
Task<Book> AddBook(string BookId, string BookName, string Author, int Quantity);

Task<Book> Update(Book model);

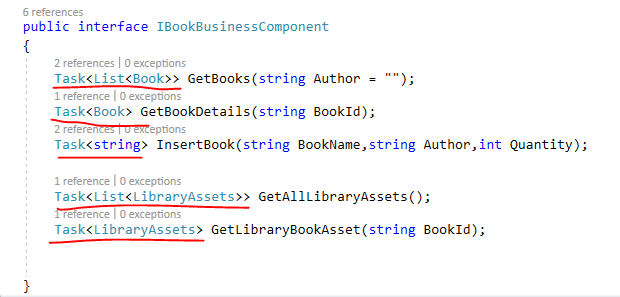
Task<Book> Delete(string BookId);

}

**BookRepository.cs in LibraryDataAccess Layer**



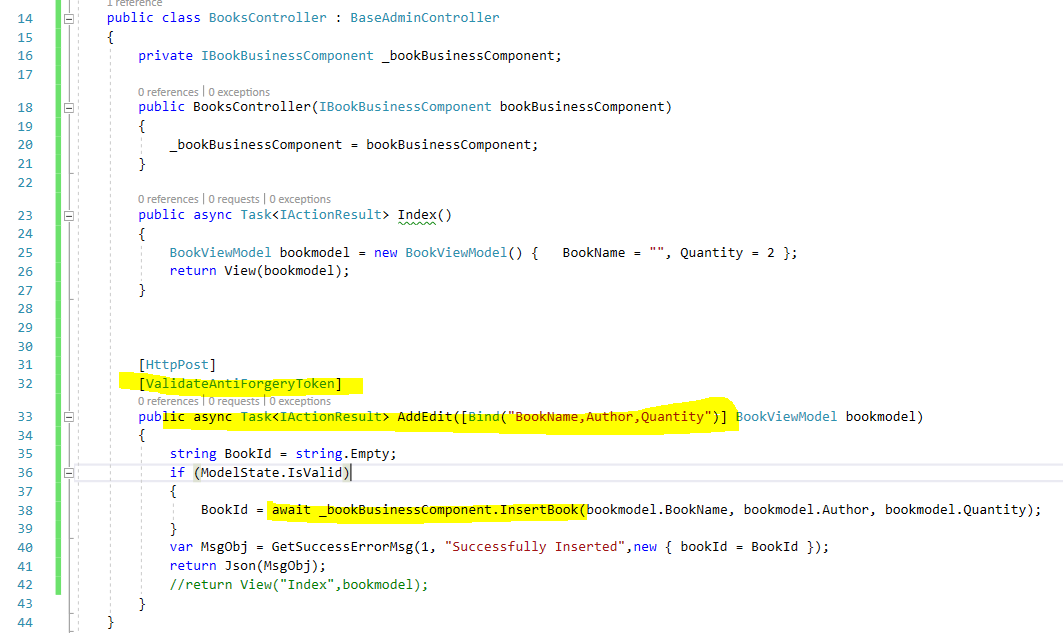
**IBookBusinessComponent.cs in LibraryBusinessComponents Layer**



**BookBusinessComponent.cs in LibraryBusinessComponents Layer**



In Controller



### **What is difference between Actionresult and IActionresult ?**

IActionResult is an interface and ActionResult is an implementation of that interface.

ActionResults is an abstract class and action results like ViewResult, PartialViewResult, JsonResult, etc., derive from ActionResult.

Let's say you want to create an action result not catered to by MVC, say an XML result.

**IActionResult is an interface, we can create a custom response as a return, when you use ActionResult you can return only predefined ones for returning a View or a resource.**

**With IActionResult we can return a response, or error as well. On the other hand, ActionResult is an abstract class, and you would need to make a custom class that inherits.**

For example see below,

1. public NoContentResult NoContentActionResult() {
2. return NoContent();
3. }
4. public IActionResult NoContentActionResult() {
5. return NoContent();
6. }
7. **OR**
9. public IActionResult JsonActionResult() {
10. return NotFound();
11. }
12. **But**
13. public JsonResult JsonActionResult() {
14. return NotFound(); **// This line will give Error**
15. }