PROGRAM.CS: namespace eBooksApp

```
{ public class Program
  { public static void Main(string[] args)
    { //CreateWebHostBuilder(args).Build().Run();
      var host = BuildWebHost(args);
using (var scope = host.Services.CreateScope())
      {var services = scope.ServiceProvider;
        try {var serviceProvider =
services.GetRequiredService<IServiceProvider>();
          var configuration =
services.GetRequiredService<IConfiguration>();
          Seed.CreateRoles(serviceProvider,
configuration).Wait();}
        catch (Exception exception)
        {var logger =
services.GetRequiredService<ILogger<Program>>();logger.LogEr
ror(exception, "An error occurred while creating roles");
        }}host.Run();
    }public static IWebHost BuildWebHost(string[] args) =>
      WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();}}
```

STARTUP.CS: namespace eBooksApp

```
{  public class Startup
  {    public Startup(IConfiguration configuration)
    {Configuration = configuration;
    }    public IConfiguration Configuration { get; }
    // This method gets called by the runtime. Use this method
to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
     services.Configure<CookiePolicyOptions>(options =>
            // This lambda determines whether user consent
for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy =
SameSiteMode.None; });
      services.AddDbContext<ApplicationDbContext>(options
=> options.UseSqlServer(
Configuration.GetConnectionString("DefaultConnection")));
      services.AddIdentity<IdentityUser,
IdentityRole>().AddDefaultUI(UIFramework.Bootstrap4)
.AddEntityFrameworkStores<ApplicationDbContext>().AddSignI
nManager<SignInManager<IdentityUser>>().AddDefaultTokenP
roviders();
      services.AddSession();
services.AddMvc().SetCompatibilityVersion(CompatibilityVersio
n.Version_2_2)
    public void Configure(IApplicationBuilder app,
IHostingEnvironment env)
    { if (env.IsDevelopment())
      {app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
      }else
      { app.UseExceptionHandler("/Home/Error");
app.UseHsts();
      }app.UseHttpsRedirection();
      app.UseStaticFiles();
      app.UseCookiePolicy();
      app.UseAuthentication();
      app.UseSession();
      app.UseMvc(routes =>
      {routes.MapRoute(
          name: "default",
          template:
"{controller=Home}/{action=Index}/{id?}");      });}}
```

WEB API:

```
    [Route("api/[controller]")]
    public class eBooksController :
ControllerBase
    {       private const string sessionId_ =
"SessionId";
        private readonly ApplicationDbContext
context_;
        private readonly IHostingEnvironment
hostingEnvironment_;
        private string webRootPath = null;
        private string ebookPath = null;
        public
eBooksController(IHostingEnvironment
hostingEnvironment, ApplicationDbContext
context)
{           context_ = context;
hostingEnvironment_ = hostingEnvironment;
webRootPath = hostingEnvironment_.WebRootPath;
ebookPath = Path.Combine(webRootPath,
"FileStorage");
}       // GET: api/<controller>
[HttpGet]
 public IEnumerable<string> Get()
{           List<string> ebooks = null;
try           {                ebooks =
Directory.GetFiles(ebookPath).ToList<string>();
for (int i = 0; i < ebooks.Count; ++i)
                    ebooks[i] =
Path.GetFileName(ebooks[i]);
           }catch{
 ebooks = new List<string>();
ebooks.Add("404 - Not Found");
           }return ebooks; }
// GET api/<controller>/5
 [HttpGet("{id}")]public async
Task<IActionResult> Download(int id)
 {List<string> ebooks = null;
string ebook = "";
try{ebooks =
Directory.GetFiles(ebookPath).ToList<string>();
if (0 <= id && id < ebooks.Count)
ebook = Path.GetFileName(ebooks[id]);
else
return NotFound();} catch {
return NotFound();
}var memory = new MemoryStream();
ebook = ebooks[id];
using (var stream = new FileStream(ebook,
FileMode.Open)){await
stream.CopyToAsync(memory);}
memory.Position = 0;
return File(memory, GetContentType(ebook),
Path.GetFileName(ebook));
}private string GetContentType(string path)
{var types = GetMimeTypes();
var ext =
Path.GetExtension(path).ToLowerInvariant();
return types[ext]; }
private Dictionary<string, string>
GetMimeTypes()
{return new Dictionary<string, string>
 {{".cs", "application/C#" },};}
// POST api/<controller>
[HttpPost]
public async Task<IActionResult> Upload(int id)
{int? publisherId_ =
HttpContext.Session.GetInt32(sessionId_);
var ebookid = context_.eBooks.Find(id);
var request = HttpContext.Request;
foreach (var ebook in request.Form.Files)
{if (ebook.Length > 0){ var path =
Path.Combine(ebookPath, ebook.FileName);
var path1 = "/FileStorage/" + ebook.FileName;
 if (ebookid != null){ ebookid.Name =
ebook.FileName;
 ebookid.Path = path1;
 try{    context_.SaveChanges();}
 catch (Exception) {}
}context_.SaveChanges();
using (var ebookStream = new FileStream(path,
FileMode.Create)){
await ebook.CopyToAsync(ebookStream);
return RedirectToAction("eBooks", "Home");
}else{return BadRequest();}}return Ok();}
```

CLIENT.CS: namespace CoreConsoleClient

```
{class CoreConsoleClient
{public HttpClient client { get; set; }
private string baseUrl_;
CoreConsoleClient(string url)
        {       baseUrl_ = url;
            client = new HttpClient();}
        public async Task<HttpResponseMessage>
SendFile(string ebookSpec, int id)
        {MultipartFormDataContent multiContent
= new MultipartFormDataContent();
  byte[] data = File.ReadAllBytes(ebookSpec);
  ByteArrayContent bytes = new
ByteArrayContent(data);
string ebookName = Path.GetFileName(ebookSpec);
multiContent.Add(bytes, "files", ebookName);
multiContent.Add(new
StringContent(id.ToString()), "id");
return await client.PostAsync(baseUrl_,
multiContent);}
public async Task<IEnumerable<string>>
GetFileList()
 {   HttpResponseMessage resp = await
client.GetAsync(baseUrl_);
      var ebooks = new List<string>();
if (resp.IsSuccessStatusCode) {
 var json = await
resp.Content.ReadAsStringAsync();
JArray jArr =
(JArray)JsonConvert.DeserializeObject(json);
foreach (var item in jArr)
{ebooks.Add(item.ToString());}} return ebooks;
} public async Task<HttpResponseMessage>
GetFile(int id)
{return await client.GetAsync(baseUrl_ + "/" +
id.ToString());}
static void Main(string[] args)
{ Console.Write("\n  CoreConsoleClient");
   if (!parseCommandLine(args))
{return;
Console.Write("Press key to start: ");
Console.ReadKey();
string url = args[0]
CoreConsoleClient client = new
CoreConsoleClient(url);
showCommandLine(args);
Console.Write("\n  sending request to {0}\n",
url);
switch (args[1])
{case "/fl": Task<IEnumerable<string>> tfl =
client.GetFileList();
var resultfl = tfl.Result;
foreach (var item in resultfl)
{ Console.Write("\n  {0}", item); } break;
case "/up": int id1 = Int32.Parse(args[3]);
Task<HttpResponseMessage> tup =
client.SendFile(args[2], id1);
Console.Write(tup.Result);break;
case "/dn":
int id = Int32.Parse(args[2]);
Task<HttpResponseMessage> tdn =
client.GetFile(id);
Console.Write(tdn.Result); break;
Console.WriteLine("\n  Press Key to exit: ");
Console.ReadKey();}}
```

SEED.CS - namespace eBooksApp.Data{

```
    public class Seed
    { public static async Task
CreateRoles(IServiceProvider serviceProvider,
IConfiguration Configuration)
{       //adding customs roles
var RoleManager =
serviceProvider.GetRequiredService<RoleManager<
IdentityRole>>();
var UserManager =
serviceProvider.GetRequiredService<UserManager<
IdentityUser>>();
var SignInManager =
serviceProvider.GetRequiredService<SignInManage
r<IdentityUser>>();
string[] roleNames = { "Admin", "User" };
IdentityResult roleResult;
foreach (var roleName in roleNames)
{//creating the roles and seeding them to the
database
var roleExist = await
RoleManager.RoleExistsAsync(roleName);
if (!roleExist) {
roleResult = await RoleManager.CreateAsync(new
IdentityRole(roleName)); }}
//creating a super user who could maintain the
web app
var poweruser = new IdentityUser
{UserName=Configuration.GetSection("AppSettings
")["UserEmail"],
Email=Configuration.GetSection("AppSettings")["
UserEmail"]};
string userPassword =
Configuration.GetSection("AppSettings")["UserPa
ssword"];
var user = await
UserManager.FindByEmailAsync(Configuration.GetS
ection("AppSettings")["UserEmail"]);
if (user == null)
{var createPowerUser = await
UserManager.CreateAsync(poweruser,
userPassword);
if (createPowerUser.Succeeded)                {
//here we tie the new user to the "Admin" role
await UserManager.AddToRoleAsync(poweruser,
"Admin");}}}}}
```

HOMECONTROLLER - namespace eBooksApp.Controllers

```
{public class HomeController : Controller{
private readonly ApplicationDbContext context_;
 private const string sessionId_ = "SessionId";
public HomeController(ApplicationDbContext
context)        {    context_ = context; }
public IActionResult Index()
 {           return
View(context_.Publishers.ToList<Publisher>());}
public IActionResult eBooks()
{           // fluent API
var ebooks = context_.eBooks.Include(l =>
l.Publishers);
var orderedLects = ebooks.OrderBy(l => l.Title)
.OrderBy(l => l.Publishers)
.Select(l => l);
 return View(orderedLects);}
[HttpGet]
[Authorize(Roles = "Admin")]
public IActionResult CreatePublisher(int id)
{ var model = new Publisher();
return View(model); }
[HttpPost]
public IActionResult CreatePublisher(int id,
Publisher crs)
 { context_.Publishers.Add(crs);
context_.SaveChanges();
return RedirectToAction("Index");        }
[HttpGet]
public IActionResult AddeBook(int id)
{ HttpContext.Session.SetInt32(sessionId_, id);
// this works too// TempData[sessionId_] = id;
```

```
 Publisher publisher =
context_.Publishers.Find(id);
 if (publisher == null)
{  return
StatusCode(StatusCodes.Status404NotFound);
}  eBook lct = new eBook() return View(lct);
}       [HttpPost]
public IActionResult AddeBook(int? id, eBook
lct) {            if (id == null)       {
return
StatusCode(StatusCodes.Status400BadRequest);
} int? publisherId_ =
HttpContext.Session.GetInt32(sessionId_);
var publisher =
context_.Publishers.Find(publisherId_);
if (publisher != null)
{if (publisher.eBooks == null)  // doesn't have
any lectures yet{ List<eBook> ebooks = new
List<eBook>();
 publisher.eBooks = ebooks; }
 publisher.eBooks.Add(lct);
try { context_.SaveChanges();}
  catch (Exception)
{    // do nothing for now} }
 return RedirectToAction("Index");}

MODEL - public class Publisher
    {   [Key]
        public int PublisherId { get; set; }
        public string Identifier { get; set; }
        public string Name { get; set; }
   public ICollection<eBook> eBooks { get; set;
}  }
   public class eBook
    {    public int eBookId { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public string Name { get; set; }
        public string Path { get; set; }
   public int? PublisherId { get; set; }
  public Publisher Publishers { get; set; }   }

APPLICATIONDB - public class
ApplicationDbContext : IdentityDbContext
    {public ApplicationDbContext() { }
   public
ApplicationDbContext(DbContextOptions<Applicati
onDbContext> options)
           : base(options)
        {        }
public DbSet<Publisher> Publishers { get; set;
}    public DbSet<eBook> eBooks { get; set; }}
```

MVC STRUCTURE - The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires.The following diagram shows the three main components and which ones reference the others: This delineation of responsibilities helps you scale the application in terms of complexity because it's easier to code, debug, and test something (model, view, or controller) that has a single job. It's more difficult to update, test, and debug code that has dependencies spread across two or more of these three areas. For example, user interface logic tends to change more frequently than business logic. If presentation code and business logic are combined in a single object, an object containing business logic must be modified every time the user interface is changed. This often introduces errors and requires the retesting of business logic after every minimal user interface change.
Model Responsibilities: The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application. Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.
View Responsibilities: Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET code in HTML markup. There should be minimal logic within views, and any logic in them should relate to presenting content. If you find the need to perform a great deal of logic in view files in order to display data from a complex model, consider using a View Component, ViewModel, or view template to simplify the view.

Controller Responsibilities: Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render (hence its name - it controls how the app responds to a given request).
What is ASP.NET Core MVC: The ASP.NET Core MVC framework is a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core.ASP.NET Core MVC provides a patterns-based way to build dynamic websites that enables a clean separation of concerns. It gives you full control over markup, supports TDD-friendly development and uses the latest web standards.
Mapping requests to responses: At its heart, ASP.NET Core apps map incoming requests to outgoing responses. At a low level, this is done with middleware, and simple ASP.NET Core apps and microservices may be comprised solely of custom middleware. When using ASP.NET Core MVC, you can work at a somewhat higher level, thinking in terms of *routes*, *controllers*, and *actions*. Each incoming request is compared with the application's routing table, and if a matching route is found, the associated action method (belonging to a controller) is called to handle the request. If no matching route is found, an error handler (in this case, returning a NotFound result) is called.ASP.NET Core MVC apps can use conventional routes, attribute routes, or both. Conventional routes are defined in code, specifying routing *conventions* using syntax like in the example below:

```
app.UseMvc(routes
=>{routes.MapRoute("default","{controller=Home}/{acti
on=Index}/{id?}");});
```

In this example, a route named "default" has been added to the routing table. It defines a route template with placeholders for *controller*, *action*, and *id*. The controller and action placeholders have default specified ("Home" and "Index", respectively), and the id placeholder is optional (by virtue of a "?" applied to it). The convention defined here states that the first part of a request should correspond to the name of the controller, the second part to the action, and then if necessary a third part will represent an id parameter. Conventional routes are typically defined in one place for the application, such as in the Configure method in the Startup class. Attribute routes are applied to controllers and actions directly, rather than specified globally. This has the advantage of making them much more discoverable when you're looking at a particular method, but does mean that routing information is not kept in one place in the application. With attribute routes, you can easily specify multiple routes for a given action, as well as combine routes between controllers and actions.

AUTHENTICATION & AUTHORIZATION - INTRODUCTION• A web application can have many end-users with different roles. • An end user is granted with access privileges based upon its role. • All these can be achieved by authenticating the user based upon login credentials. • In the following slides, we are going to explore how a user can be authenticated and how the authorization works depending upon the role in an ASP.Net Core Web App. AUTHENTICATION: IT IS A PROCESS OF VALIDATING A USER'S CREDENTIALS AGAINST THE STORED VALUES IN THE DATABASE OR OTHER SOURCES. ROLES: THERE CAN BE DIFFERENT END USERS OF A WEB APPLICATION WITH DIFFERENT ROLES. EXAMPLE: ADMINISTRATOR, REGISTERED USER OR GUEST ETC. AUTHORIZATION: IT IS A PROCESS TO IDENTIFY THE PRIVILEGES OF THE USER BASED UPON ITS ROLE. Authentication: • Authentication in ASP.Net Core applications is provided by ASP.NET Core Identity. (There are some other third party services that does the same) • ASP.NET Core Identity is a membership or identity management system that comes with ASP.NET Core web development stack. • Some of the facilities pro vided by ASP.NET Identity are user registration, login etc. AUTHORIZATION: services.AddAuthorization() in startup.cs is used to implement Authorizartion services. Authorization or access privileges are checked in two ways; Role Based, Policy Based. ROLE BASED AUTH: "Authorize" attribute is used to define authority of a user with a given role to access controller method. Eg. [Authorize(Roles = "User")] The above usage of attribute gives access to the following controller method to only the user with role as "User". POLICY BASED AUTH: Policy based authorization requires the user to adhere the defined policy in order to get access to a controller method. Eg.

```
Services.AddAuthorization(options=>{
options.AddPolicy("OnlyAdminAccess", policy =>
policy.RequireRole("Admin"));});

[Authorize(Policy = "OnlyAdminAccess")] public IActionResult
PolicyExample() { ViewData["role"] = "Admin"; return
View("Test");
```