Laborator 5 – Aplicatii Web cu ASP.NET Core si Entity Framework Code First

1. In laboratorul curent, vom dezvolta aplicatia web creata la laboratorul anterior. Deschidem Visual Studio si apoi alegem optiunea **Open a project or solution** si cautam proiectul **Nume_Pren_Lab2,** creat anterior. Pentru a putea fi evaluata activitatea aferenta fiecarui laborator, laboratorul curent il vom dezvolta pe un branch nou, diferit de cel master care se afla deja creat. Utilizand pasii indicati in Lab 1, pct. 22 vom crea un branch nou (bazat pe lab 4) pe care il vom denumi Laborator5
2. In laboratorul curent vom realiza functionalitatile care permit realizarea de imprumuri pentru cartile existente in biblioteca. Astfel vom adauga o noua clasa model va reprezenta membri abonati la biblioteca. Apasam click dreapta pe numele folderului Models->Add->New Item alegem Class si vom denumi fisierul Member.cs. In acest fisier se va genera automat o clasa cu denumirea Member pentru care adaugam proprietatile de mai jos.

```
public class Member
 {

        public int ID { get; set; }

        public string? FirstName { get; set; }

        public string? LastName { get; set; }

        public string? Adress { get; set; }

        public string Email { get; set; }

        public string? Phone { get; set; }

        [Display(Name = "Full Name")]
        public string? FullName
        {
            get
            {
                return FirstName + " " + LastName;
            }
        }
        public ICollection<Borrowing>? Borrowings { get; set; }
 }
```

3. Pentru a putea modela colectia de carti imprumutate de un membru vom crea o noua clasa cu denumirea Borrowing, urmand pasii indicati la punctul anterior (click dreapta pe numele folderului Models->Add->New Item). Pentru fiecare membru vom putea pastra o colectie de carti imprumutate si data la care trebuie returnate fiecare dintre acestea.

```
public class Borrowing
    {
        public int ID { get; set; }

        public int? MemberID { get; set; }
        public Member? Member { get; set; }

        public int? BookID { get; set; }
```

```
        public Book? Book { get; set; }

        [DataType(DataType.Date)]
        public DateTime ReturnDate { get; set; }
    }
```

4. Modelăm legatura cu entitatea Borrowing adaugand în Book:

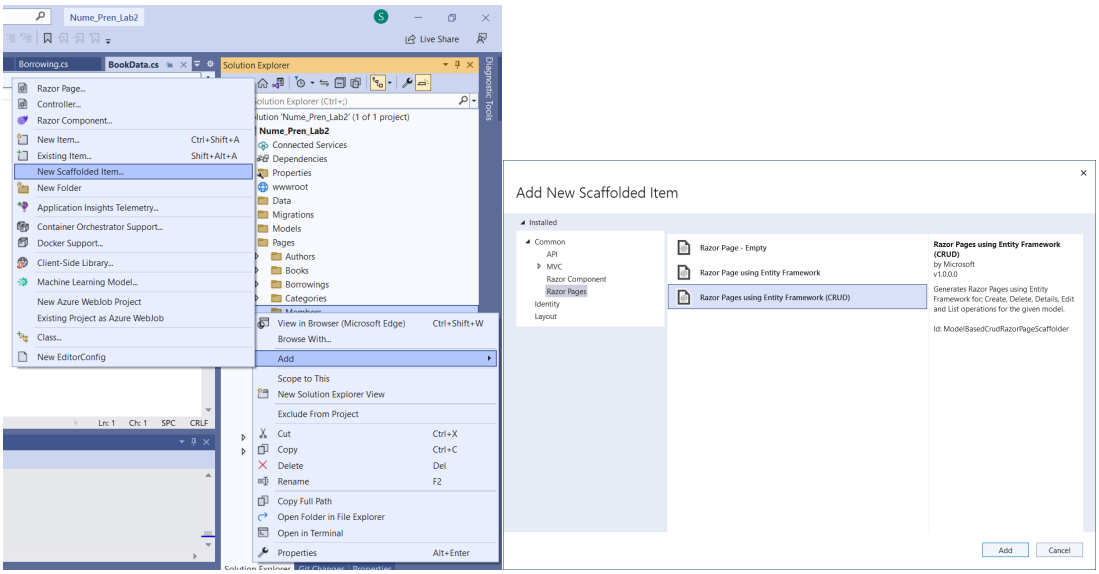
```
public class Book
{
    public int ID { get; set; }

    [Display(Name = "Book Title")]

    public string Title { get; set; }

    [Column(TypeName = "decimal(6, 2)")]
    public decimal Price { get; set; }

    [DataType(DataType.Date)]
    public DateTime PublishingDate { get; set; }

    public int? AuthorID { get; set; }
    public Author? Author { get; set; }
    public int? PublisherID { get; set; }
    public Publisher? Publisher { get; set; }

    public ICollection<Borrowing>? Borrowings { get; set; }
    public ICollection<BookCategory>? BookCategories { get; set; }

}
```

5. Vom migra modificarile aduse asupra modelului in baza de date. Deschidem Package Manage Console si rulam urmatoarele doua comenzi:

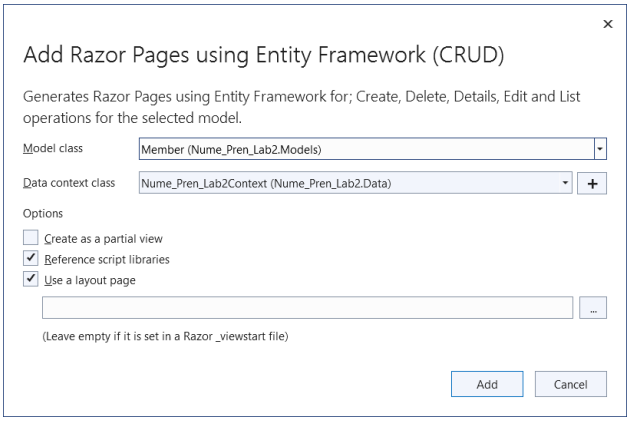
   Add-migration Borrowings

   Update-database

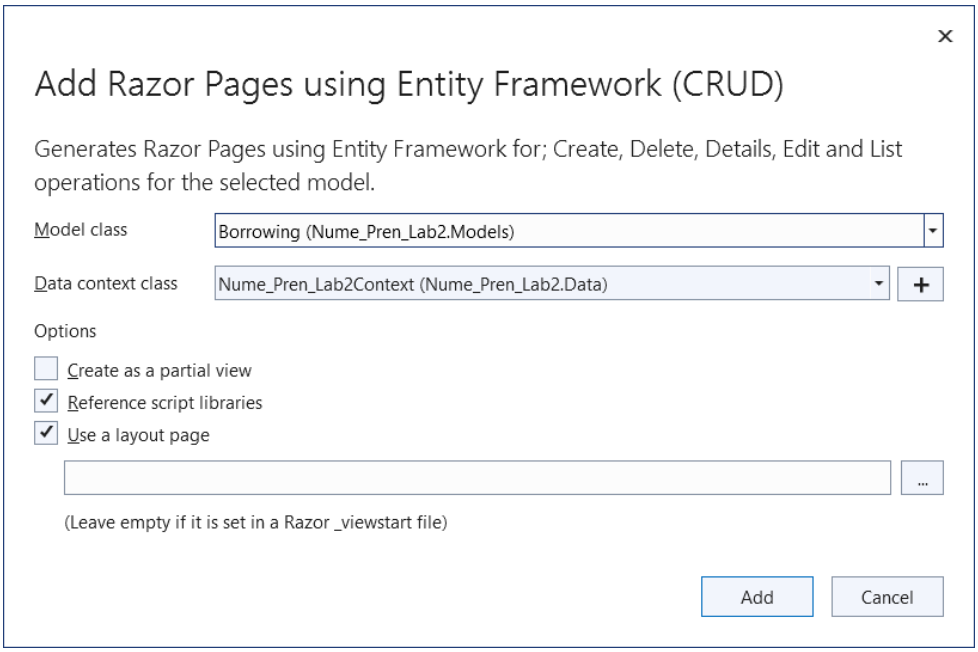
6. Generam paginile asociate entitatii nou create Member. In folderul Pages cream un nou folder cu denumirea Members apoi apasam click-dreapta pe numele folderului Members->Add->New Scaffolded Item



7. Selectam clasa model Member si apoi apasam butonul Add



8. Pentru a genera paginile asociate entitatii Borrowing, in folderul Pages cream un nou folder cu denumirea Borrowings apoi apasam click-dreapta pe numele folderului Borrowings->Add->New Scaffolded Item. Selectam clasa model Borrowing si apoi apasam butonul Add.



9. Adaugam codul de mai jos in fisierul _Layout.cshtml astfel incat sa putem accesa in bara de navigare Members si Borrowings:

```
  <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Authors/Index">Authors</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Publishers/Index">Publishers</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Categories/Index">Categories</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Members/Index">Members</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Borrowings/Index">Borrowings</a>
                        </li>
                    </ul>
                </div>
```

10. Daca rulam aplicatia, navigam la pagina pagina \Borrowings\Create si incarcam sa cream un imprumut, observam ca in cele doua liste derulante aferente membrilor si cartilor se

afiseaza id-urile acestora. Pentru a se afisa datele cartii si numele abonatului (in loc de id-uri) vom modifica metoda OnGet() din fisierul Pages\Borrowings\Create.cshtml.cs astfel:

```
    public IActionResult OnGet()
            {
                    var bookList = _context.Book
                     .Include(b=> b.Author)
                     .Select(x => new
                {
                    x.ID,
                    BookFullName = x.Title+" - "+x.Author.LastName + " " +
    x.Author.FirstName
                });


                ViewData["BookID"] = new SelectList(bookList, "ID",
    "BookFullName");
                ViewData["MemberID"] = new SelectList(_context.Member, "ID",
    "FullName");
                return Page();
            }
```

11. Modificam fisierul Pages\Borrowings\Index.cshtml.cs, metoda OnGetAsync(). Modificam interogarea LINQ pentru a aduce datele referitoare la autor din entitatea Author pentru a putea afisa numele si prenumele autorului

```
    public async Task OnGetAsync()
            {
                if (_context.Borrowing != null)
                {
                    Borrowing = await _context.Borrowing
                    .Include(b => b.Book)
                        .ThenInclude(b => b.Author)
                    .Include(b => b.Member).ToListAsync();
                }
            }
```

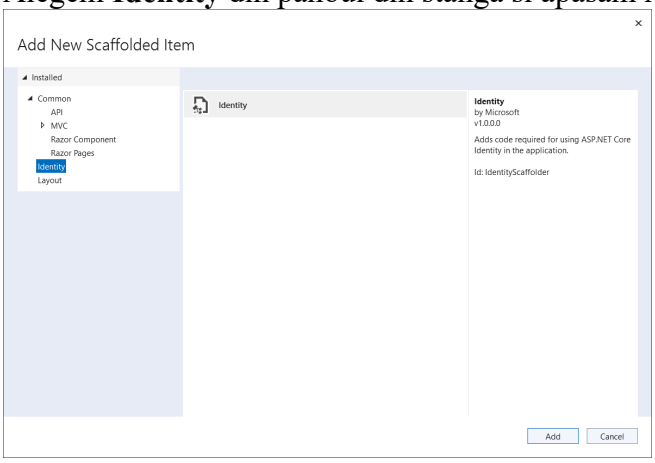
12. Modificam si modul de afisare a datelor in fisierul Pages\Borrowings\Index.cshtml astfel:

```
<thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Borrowing[0].Member)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Borrowing[0].Book)
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Borrowing[0].Book.Author)
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Borrowing[0].ReturnDate)
            </th>
            <th></th>
        </tr>
    </thead>
```

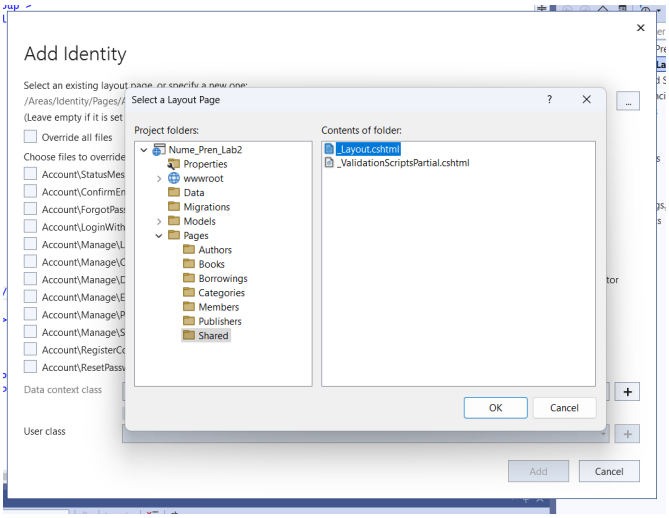
```
 <tbody>
@foreach (var item in Model.Borrowing) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Member.FullName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Book.Title)
            </td>
                <td>
                    @Html.DisplayFor(modelItem =>
item.Book.Author.FullName)
                </td>
            <td>
                @Html.DisplayFor(modelItem => item.ReturnDate)
            </td>
            <td>
                <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-page="./Details" asp-route-id="@item.ID">Details</a>
|
                <a asp-page="./Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
    </tbody>

….
```

13. Pentru a realiza autentificarea si autorizarea vom folosi libraria de clase Razor **Identity** din ASP.NET Core. Pentru a adauga libraria, din Solution Explorer, faceți clic dreapta pe proiect > Add > New Scaffolded Item...

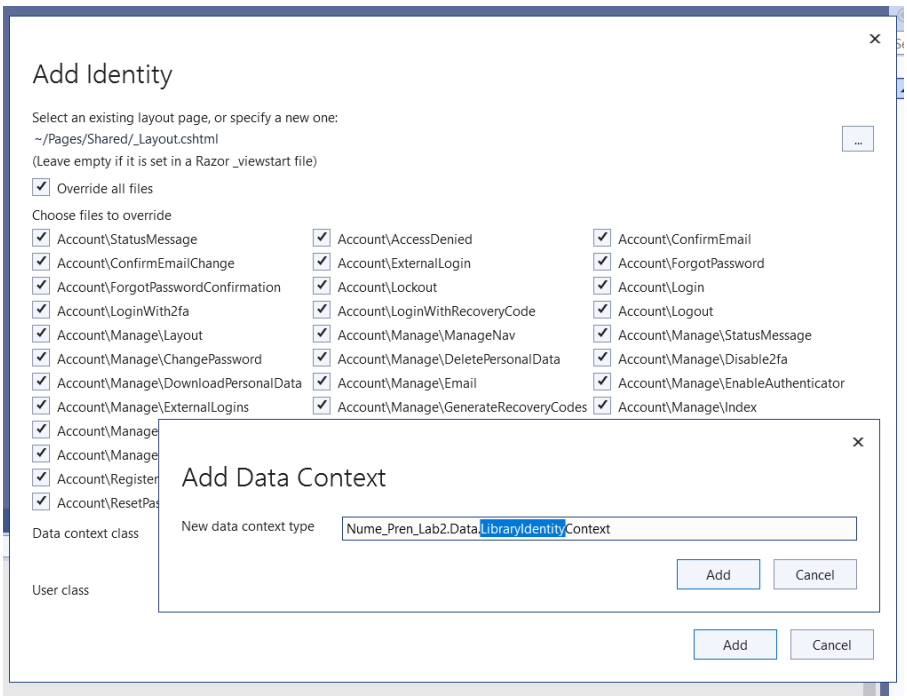
    Alegem **Identity** din panoul din stanga si apasam Add



14. În fereasta Add Identity, selectați pagina de layout existentă: ~/Pages/Shared/_Layout.cshtml



15. Bifam Override all Files si cream o noua clasa LibraryIdentityContext pe care o vom utiliza pentru gestiunea autentificarii si autorizarii si apoi apasam butonul Add



Observam ca in Solution Explorer a fost creat un folder cu denumirea Areas si un un subfolder cu denumirea Identity.

16. In fisierul Program.cs inregistram nou clasa LibraryIdentityContext adaugand codul de mai jos :

```
…
    builder.Services.AddRazorPages();
    builder.Services.AddDbContext<Nume_Pren_Lab2Context>(options =>
```

```
    options.UseSqlServer(builder.Configuration.GetConnectionString("Nume_Pre
    n_Lab2Context") ?? throw new InvalidOperationException("Connection
    string 'Nume_Pren_Lab2Context' not found.")));

    builder.Services.AddDbContext<LibraryIdentityContext>(options =>

    options.UseSqlServer(builder.Configuration.GetConnectionString("Nume_Pre
    n_Lab2Context") ?? throw new InvalidOperationException("Connection
    string 'Nume_Pren_Lab2Context' not found.")));


    builder.Services.AddDefaultIdentity<IdentityUser>(options =>
    options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<LibraryIdentityContext>();

    var app = builder.Build();
    …
```

17. Vom migra modelul utilizat de Identity in baza de date. Pentru ca acum avem doua clase de tip Context trebuie sa precizam in mod explicit pe care dorim sa utilizam. Deschidem Package Manage Console si rulam urmatoarele doua comenzi:

```
Add-migration  CreateIdentity -Context LibraryIdentityContext

Update-database -Context LibraryIdentityContext
```

18. Pentru ca in coltul dreapta sus sa ne apara functionalitatile de logare utilizator vom adauga in fisierul __Layout.cshtml referinta la view-ul partial _LoginPartial (care a fost generat de Identity):

```
….
  </li>
                          <li class="nav-item">
                              <a class="nav-link text-dark" asp-area="" asp-
page="/Borrowings/Index">Borrowings</a>
                          </li>

                      </ul>
                      <partial name="_LoginPartial" />
                  </div>
              </div>
          </nav>
      </header>
…..
```

19. Dorim ca atunci cand se inregistreaza un utilizator nou in aplicatie adresa acestuia de email sa fie salvata concomitent si in tabelul Member, iar ulterior un angajat al bibliotecii ii va completa celelalte date de contact. In fisierul Areas/Identity/Pages/Account/Register.cshtml.cs adaugam in clasa RegisterModel:

```
    ……
    public class RegisterModel : PageModel
        {
            private readonly SignInManager<IdentityUser> _signInManager;
            private readonly UserManager<IdentityUser> _userManager;
```

```
            private readonly IUserStore<IdentityUser> _userStore;
            private readonly IUserEmailStore<IdentityUser> _emailStore;
            private readonly ILogger<RegisterModel> _logger;
            private readonly IEmailSender _emailSender;

            private readonly Nume_Pren_Lab2.Data.Nume_Pren_Lab2Context
    _context;



            public RegisterModel(
                UserManager<IdentityUser> userManager,
                IUserStore<IdentityUser> userStore,
                SignInManager<IdentityUser> signInManager,
                ILogger<RegisterModel> logger,
                IEmailSender emailSender,
                Nume_Pren_Lab2.Data.Nume_Pren_Lab2Context context)
            {
                _userManager = userManager;
                _userStore = userStore;
                _emailStore = GetEmailStore();
                _signInManager = signInManager;
                _logger = logger;
                _emailSender = emailSender;
                _context = context;
            }
            [BindProperty]
            public Member Member { get; set; }

            [BindProperty]
            public InputModel Input { get; set; }


    ......
```

20. In acceasi clasa modificam continutul metodei OnPostAsync :

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
             returnUrl ??= Url.Content("~/");
             ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

                var user = CreateUser();

                await _userStore.SetUserNameAsync(user, Input.Email,
CancellationToken.None);
                await _emailStore.SetEmailAsync(user, Input.Email,
CancellationToken.None);
                var result = await _userManager.CreateAsync(user,
Input.Password);

                Member.Email = Input.Email;
                _context.Member.Add(Member);
                await _context.SaveChangesAsync();

                if (result.Succeeded)
                {
```

```
                    _logger.LogInformation("User created a new account with
password.");

                    var userId = await _userManager.GetUserIdAsync(user);
                    var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                    code =
WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                    var callbackUrl = Url.Page(
                        "/Account/ConfirmEmail",
                        pageHandler: null,
                        values: new { area = "Identity", userId = userId,
code = code, returnUrl = returnUrl },
                        protocol: Request.Scheme);

                    await _emailSender.SendEmailAsync(Input.Email, "Confirm
your email",
                        $"Please confirm your account by <a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");



                    if
(_userManager.Options.SignIn.RequireConfirmedAccount)
                    {
                        return RedirectToPage("RegisterConfirmation", new {
email = Input.Email, returnUrl = returnUrl });
                    }
                    else
                    {
                        await _signInManager.SignInAsync(user,
isPersistent: false);
                        return LocalRedirect(returnUrl);
                    }

                }
  return Page();
        }
```

21. In fisierul Members\Index.cshtml vom sterge link-ul la pagina Members\Create si link-ul la pagina Members\Delete:

```
    @page
    @model Nume_Pren_Lab2.Pages.Members.IndexModel

    @{
        ViewData["Title"] = "Index";
        Layout = "~/Pages/Shared/_Layout.cshtml";
    }

    <h1>Index</h1>

    <p>
        <a asp-page="Create">Create New</a>
    </p>
    <table class="table">
        <thead>
```

```
            <tr>
                <th>
                    @Html.DisplayNameFor(model => model.Member[0].FirstName)
                </th>
                <th>
   ….

</td>
             <td>
                 <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                 <a asp-page="./Details" asp-route-id="@item.ID">Details</a>
|
                 <a asp-page="./Delete" asp-route-id="@item.ID">Delete</a>
             </td>
            </tr>
```

Rulam aplicatia si verificam noile functionalitati adaugate: crearea conturi de utilizatori, logarea in cont , crearea de imprumuturi pentru membri.

Sarcina de laborator:
1.Modificati pagina Borrowings/Edit astfel incat in listele derulante sa apara numele membrilor si detaliile cartilor (in prezent sunt afisate id-urile acestora)

2. Modificati paginile Details si Delete din folderul Borrowings astfel incat pe acestea sa fie afisate numele membrilor si detaliile cartii imprumutate.