Full .NET 8 MVC + Web API + Swagger + Razor-Fetch Integration

This package includes **complete backend + frontend (Razor) code** wired end-to-end: - **Auth API** (signup/login/logout) issuing JWT into **HttpOnly cookie** `access_token` - **Employees / Products / Orders** REST APIs with **pagination + search** and **DataAnnotations** validation - **Swagger/OpenAPI** docs (cookie & Bearer auth) - **Razor views using** `fetch()` to call the APIs directly (no Swagger UI needed for the app pages) - **SQL Server schema** (SSMS-ready)

> Replace your duplicates (`OrderItem`, `Product`) and fix the stray `[Authorize] git add;` line as shown. You can drop these files into a clean repo (or overwrite where appropriate).

File Tree (suggested)

```
EmployeeCrudPdf/
├── Controllers/
│   ├── AuthApiController.cs
│   ├── EmployeesApiController.cs
│   ├── ProductsApiController.cs
│   ├── OrdersApiController.cs
│   ├── EmployeesController.cs          # (optional legacy MVC detail views)
│   ├── ProductsController.cs           # fix [Authorize] line
│   └── OrdersController.cs             # (optional legacy MVC detail views)
├── Data/
│   ├── EmployeeRepository.cs
│   ├── ProductRepository.cs
│   ├── OrderRepository.cs
│   ├── SqlConnectionFactory.cs
│   ├── IEmployeeRepository.cs
│   ├── IProductRepository.cs
│   ├── IOrderRepository.cs
│   ├── ISqlConnectionFactory.cs
│   └── UserRepository.cs
├── Dtos/
│   ├── EmployeeDtos.cs
│   ├── ProductDtos.cs
│   └── OrderDtos.cs
├── Exceptions/
│   ├── AppException.cs
│   ├── DatabaseException.cs
```

```
│   ├── DuplicateEntityException.cs
│   └── NotFoundException.cs
├── Middleware/
│   └── ExceptionHandlingMiddleware.cs
├── Models/
│   ├── Employee.cs
│   ├── Product.cs
│   ├── Order.cs
│   ├── OrderItem.cs
│   ├── AddToOrderViewModel.cs
│   ├── LoginRegisterViewModels.cs
│   └── User.cs
├── Services/
│   ├── IJwtService.cs
│   └── JwtService.cs
├── Services/Logging/
│   ├── IAppLogger.cs
│   └── FileLogger.cs
├── Utilities/
│   └── LinqPagingExtensions.cs
├── Views/
│   ├── Shared/_Layout.cshtml
│   ├── ApiFront/
│   │   ├── Index.cshtml                # landing with links to the API-driven
pages
│   │   ├── Employees.cshtml            # full CRUD via fetch => /api/employees
│   │   ├── Products.cshtml             # full CRUD via fetch => /api/products
│   │   └── Orders.cshtml               # create/list/view items via fetch => /
api/orders
│   └── AccountMvc/
│       ├── Login.cshtml                # optional classic forms
│       └── Register.cshtml
├── wwwroot/
│   ├── css/site.css
│   └── js/api-helpers.js
├── appsettings.json
├── EmployeeCrudPdf.csproj
├── Program.cs
├── HttpContextUserExtensions.cs
└── README.md
```

Backend

Program.cs

```
using System.Reflection;
using System.Text;
using EmployeeCrudPdf.Data;
using EmployeeCrudPdf.Middleware;
using EmployeeCrudPdf.Services;
using EmployeeCrudPdf.Services.Logging;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Config
var jwtCfg = builder.Configuration.GetSection("Jwt");

// MVC + Razor
builder.Services.AddControllersWithViews().AddRazorRuntimeCompilation();

// DI
builder.Services.AddSingleton<IAppLogger, FileLogger>();
builder.Services.AddSingleton<IJwtService, JwtService>();
builder.Services.AddScoped<ISqlConnectionFactory, SqlConnectionFactory>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IEmployeeRepository, EmployeeRepository>();
builder.Services.AddScoped<IProductRepository, ProductRepository>();
builder.Services.AddScoped<IOrderRepository, OrderRepository>();

// Session
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(o =>
{
    o.Cookie.Name = ".EmployeeCrudPdf.Session";
    o.IdleTimeout = TimeSpan.FromHours(8);
    o.Cookie.HttpOnly = true;
    o.Cookie.IsEssential = true;
});

// Auth: Cookies for MVC; JWT for APIs (reads token from 'access_token' cookie)
builder.Services
    .AddAuthentication(options =>
    {
```

```
        options.DefaultAuthenticateScheme =
CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme    =
CookieAuthenticationDefaults.AuthenticationScheme;
    })
    .AddCookie(options =>
    {
        options.LoginPath = "/AccountMvc/Login";
        options.LogoutPath = "/AccountMvc/Logout";
        options.SlidingExpiration = true;
        options.ExpireTimeSpan = TimeSpan.FromHours(8);
        options.Cookie.HttpOnly = true;
        options.Cookie.SameSite = SameSiteMode.Lax;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtCfg["Issuer"],
            ValidAudience = jwtCfg["Audience"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtCfg["Key"]!)),
            ClockSkew = TimeSpan.FromSeconds(30)
        };
        options.Events = new JwtBearerEvents
        {
            OnMessageReceived = ctx =>
            {
                if (string.IsNullOrEmpty(ctx.Token) &&
ctx.Request.Cookies.TryGetValue("access_token", out var token))
                    ctx.Token = token;
                return Task.CompletedTask;
            }
        };
    });

builder.Services.AddAuthorization();

// Swagger
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "EmployeeCrudPdf API",
```

```
        Version = "v1",
        Description =
"Auth + Employees + Products + Orders. Pagination + search (LINQ) &
DataAnnotations validation."
    });

    var xmlName = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlName);
    if (File.Exists(xmlPath)) c.IncludeXmlComments(xmlPath,
includeControllerXmlComments: true);

    c.AddSecurityDefinition("cookieAuth", new OpenApiSecurityScheme
    {
        Type = SecuritySchemeType.ApiKey,
        In = ParameterLocation.Cookie,
        Name = "access_token",
        Description = "JWT stored in HttpOnly 'access_token' cookie",
    });
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = "Bearer {token}",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Scheme = "bearer",
        BearerFormat = "JWT"
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement {
        { new OpenApiSecurityScheme { Reference = new OpenApiReference { Type =
ReferenceType.SecurityScheme, Id = "cookieAuth" } }, Array.Empty<string>() },
        { new OpenApiSecurityScheme { Reference = new OpenApiReference { Type =
ReferenceType.SecurityScheme, Id = "Bearer" } }, Array.Empty<string>() }
    });
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(o =>
    {
        o.DocumentTitle = "EmployeeCrudPdf API Docs";
        o.DisplayRequestDuration();
        o.EnablePersistAuthorization();
    });
}
```

```
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseSession();
app.UseAuthentication();
app.UseAuthorization();
app.UseMiddleware<ExceptionHandlingMiddleware>();

// Default route: landing page with links to API-driven Razor views
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=ApiFront}/{action=Index}/{id?}");

// Attribute-routed APIs
app.MapControllers();

app.Run();
```

Controllers/AuthApiController.cs

```
using System.ComponentModel.DataAnnotations;
using System.Security.Claims;
using EmployeeCrudPdf.Data;
using EmployeeCrudPdf.Models;
using EmployeeCrudPdf.Services;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EmployeeCrudPdf.Controllers
{
    public class AuthLoginDto
    {
        [Required] public string UsernameOrEmail { get; set; } = "";
        [Required] public string Password { get; set; } = "";
    }
    public class AuthRegisterDto
    {
        [Required, StringLength(80)] public string Username { get; set; } = "";
        [Required, EmailAddress, StringLength(200)] public string Email { get;
set; } = "";
        [Required, MinLength(6)] public string Password { get; set; } = "";
    }

    /// <summary>Auth endpoints: signup, login, logout. Issues JWT into HttpOnly
```

```
cookie 'access_token'.</summary>
    [ApiController]
    [Route("api/auth")]
    public class AuthApiController : ControllerBase
    {
        private readonly IUserRepository _users;
        private readonly IJwtService _jwt;
        public AuthApiController(IUserRepository users, IJwtService jwt) {
_users = users; _jwt = jwt; }

        /// <summary>Registers a new user and signs them in.</summary>
        [HttpPost("signup")]
        [AllowAnonymous]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status409Conflict)]
        public async Task<IActionResult> Signup([FromBody] AuthRegisterDto dto)
        {
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
            var byUsername = await
_users.GetByUsernameOrEmailAsync(dto.Username);
            var byEmail = await _users.GetByUsernameOrEmailAsync(dto.Email);
            if (byUsername != null || byEmail != null) return Conflict("User
already exists");

            var user = new User
            {
                Username = dto.Username,
                Email = dto.Email,
                PasswordHash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
                CreatedAt = DateTime.UtcNow
            };
            user.Id = await _users.CreateAsync(user);

            await SignInAndSetJwt(user);
            return NoContent();
        }

        /// <summary>Logs in a user.</summary>
        [HttpPost("login")]
        [AllowAnonymous]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public async Task<IActionResult> Login([FromBody] AuthLoginDto dto)
        {
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
            var user = await
_users.GetByUsernameOrEmailAsync(dto.UsernameOrEmail);
            if (user == null || !BCrypt.Net.BCrypt.Verify(dto.Password,
```

```
user.PasswordHash))
                return Unauthorized("Invalid credentials");

            await SignInAndSetJwt(user);
            return NoContent();
        }

        /// <summary>Logs out the current user (clears cookie & session).</
summary>
        [HttpPost("logout")]
        [Authorize]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        public async Task<IActionResult> Logout()
        {
            HttpContext.Session.Clear();
            if (Request.Cookies.ContainsKey("access_token"))
Response.Cookies.Delete("access_token");
            await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return NoContent();
        }

        private async Task SignInAndSetJwt(User user)
        {
            var claims = new List<Claim>
            {
                new(ClaimTypes.NameIdentifier, user.Id.ToString()),
                new(ClaimTypes.Name, user.Username),
                new(ClaimTypes.Email, user.Email)
            };
            await HttpContext.SignInAsync(
                CookieAuthenticationDefaults.AuthenticationScheme,
                new ClaimsPrincipal(new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme)),
                new AuthenticationProperties { IsPersistent = true, ExpiresUtc
= DateTimeOffset.UtcNow.AddHours(8) });

            var token = _jwt.CreateToken(user);
            Response.Cookies.Append("access_token", token, new CookieOptions
            {
                HttpOnly = true,
                Secure = false, // set true on HTTPS
                SameSite = SameSiteMode.Lax,
                Expires = DateTimeOffset.UtcNow.AddHours(8),
                IsEssential = true
            });
        }
```

```
    }
}
```

Controllers/EmployeesApiController.cs

```
using EmployeeCrudPdf.Data;
using EmployeeCrudPdf.Dtos;
using EmployeeCrudPdf.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Mvc;

namespace EmployeeCrudPdf.Controllers
{
    /// <summary>Employee endpoints (JWT required). Supports pagination &
search.</summary>
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("api/employees")]
    public class EmployeesApiController : ControllerBase
    {
        private readonly IEmployeeRepository _repo;
        public EmployeesApiController(IEmployeeRepository repo) => _repo = repo;

        /// <summary>Gets a paged list of employees for the current user.</
summary>
        [HttpGet]
        [ProducesResponseType(typeof(PagedResult<EmployeeReadDto>),
StatusCodes.Status200OK)]
        public async Task<IActionResult> List([FromQuery] string? q,
[FromQuery] int page = 1, [FromQuery] int pageSize = 10)
        {
            var uid = HttpContext.RequireUserId();
            var rows = await _repo.GetAllAsync(uid, q, page, pageSize);
            var total = await _repo.CountAsync(uid, q);

            var dto = new PagedResult<EmployeeReadDto>
            {
                Items = rows.Select(e => new EmployeeReadDto { Id = e.Id, Name
= e.Name, Department = e.Department, Email = e.Email, Salary = e.Salary }),
                TotalCount = total, Page = page, PageSize = pageSize
            };
            return Ok(dto);
        }

        /// <summary>Creates an employee.</summary>
```

```
        [HttpPost]
        [ProducesResponseType(typeof(EmployeeReadDto),
StatusCodes.Status201Created)]
        public async Task<IActionResult> Create([FromBody] EmployeeCreateDto
dto)
        {
            var uid = HttpContext.RequireUserId();
            var id = await _repo.CreateAsync(uid, new Employee { Name =
dto.Name, Department = dto.Department, Email = dto.Email, Salary =
dto.Salary });
            var created = await _repo.GetByIdAsync(uid, id);
            var read = new EmployeeReadDto { Id = created.Id, Name =
created.Name, Department = created.Department, Email = created.Email, Salary =
created.Salary };
            return CreatedAtAction(nameof(Get), new { id = read.Id }, read);
        }

        /// <summary>Gets an employee by id.</summary>
        [HttpGet("{id:int}")]
        [ProducesResponseType(typeof(EmployeeReadDto), StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Get(int id)
        {
            var uid = HttpContext.RequireUserId();
            var e = await _repo.GetByIdAsync(uid, id);
            return Ok(new EmployeeReadDto { Id = e.Id, Name = e.Name,
Department = e.Department, Email = e.Email, Salary = e.Salary });
        }

        /// <summary>Updates an employee.</summary>
        [HttpPut("{id:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Update(int id, [FromBody]
EmployeeCreateDto dto)
        {
            var uid = HttpContext.RequireUserId();
            await _repo.UpdateAsync(uid, new Employee { Id = id, Name =
dto.Name, Department = dto.Department, Email = dto.Email, Salary =
dto.Salary });
            return NoContent();
        }

        /// <summary>Deletes an employee.</summary>
        [HttpDelete("{id:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Delete(int id)
```

```
        {
            var uid = HttpContext.RequireUserId();
            await _repo.DeleteAsync(uid, id);
            return NoContent();
        }
    }
}
```

Controllers/ProductsApiController.cs

```
using EmployeeCrudPdf.Data;
using EmployeeCrudPdf.Dtos;
using EmployeeCrudPdf.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Mvc;

namespace EmployeeCrudPdf.Controllers
{
    /// <summary>Product endpoints (JWT required). Supports pagination &
search.</summary>
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("api/products")]
    public class ProductsApiController : ControllerBase
    {
        private readonly IProductRepository _repo;
        public ProductsApiController(IProductRepository repo) => _repo = repo;

        /// <summary>Gets a paged list of products for the current user.</
summary>
        [HttpGet]
        [ProducesResponseType(typeof(PagedResult<ProductReadDto>),
StatusCodes.Status200OK)]
        public async Task<IActionResult> List([FromQuery] string? q,
[FromQuery] int page = 1, [FromQuery] int pageSize = 10)
        {
            var uid = HttpContext.RequireUserId();
            var rows = await _repo.GetAllAsync(uid, q, page, pageSize);
            var total = await _repo.CountAsync(uid, q);
            var dto = new PagedResult<ProductReadDto>
            {
                Items = rows.Select(p => new ProductReadDto { Id = p.Id, Name =
p.Name, Price = p.Price, Category = p.Category }),
                TotalCount = total, Page = page, PageSize = pageSize
            };
```

```
            return Ok(dto);
        }

        /// <summary>Creates a product.</summary>
        [HttpPost]
        [ProducesResponseType(typeof(ProductReadDto),
StatusCodes.Status201Created)]
        public async Task<IActionResult> Create([FromBody] ProductCreateDto dto)
        {
            var uid = HttpContext.RequireUserId();
            var id = await _repo.CreateAsync(uid, new Product { Name =
dto.Name, Price = dto.Price, Category = dto.Category });
            var p = await _repo.GetByIdAsync(uid, id);
            return CreatedAtAction(nameof(Get), new { id = p.Id }, new
ProductReadDto { Id = p.Id, Name = p.Name, Price = p.Price, Category =
p.Category });
        }

        /// <summary>Gets a product by id.</summary>
        [HttpGet("{id:int}")]
        [ProducesResponseType(typeof(ProductReadDto), StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Get(int id)
        {
            var uid = HttpContext.RequireUserId();
            var p = await _repo.GetByIdAsync(uid, id);
            return Ok(new ProductReadDto { Id = p.Id, Name = p.Name, Price =
p.Price, Category = p.Category });
        }

        /// <summary>Updates a product.</summary>
        [HttpPut("{id:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Update(int id, [FromBody]
ProductCreateDto dto)
        {
            var uid = HttpContext.RequireUserId();
            await _repo.UpdateAsync(uid, new Product { Id = id, Name =
dto.Name, Price = dto.Price, Category = dto.Category });
            return NoContent();
        }

        /// <summary>Deletes a product.</summary>
        [HttpDelete("{id:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Delete(int id)
```

```
        {
            var uid = HttpContext.RequireUserId();
            await _repo.DeleteAsync(uid, id);
            return NoContent();
        }
    }
}
```

Controllers/OrdersApiController.cs

```
using EmployeeCrudPdf.Data;
using EmployeeCrudPdf.Dtos;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Mvc;

namespace EmployeeCrudPdf.Controllers
{
    /// <summary>Order endpoints (JWT required). Create orders and manage
items.</summary>
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("api/orders")]
    public class OrdersApiController : ControllerBase
    {
        private readonly IOrderRepository _orders;
        private readonly IProductRepository _products;

        public OrdersApiController(IOrderRepository orders, IProductRepository
products)
        { _orders = orders; _products = products; }

        /// <summary>Gets a paged list of orders for the current user.</summary>
        [HttpGet]
        [ProducesResponseType(typeof(PagedResult<OrderListItemDto>),
StatusCodes.Status200OK)]
        public async Task<IActionResult> List([FromQuery] string? q,
[FromQuery] int page = 1, [FromQuery] int pageSize = 10)
        {
            var uid = HttpContext.RequireUserId();
            var (items, total) = await _orders.ListAsync(uid, q, page,
pageSize);
            var dto = new PagedResult<OrderListItemDto>
            {
                Items = items.Select(o => new OrderListItemDto
                { Id = o.Id, OrderNo = o.OrderNo, CreatedAt = o.CreatedAt,
```

```
Total = o.Total, ItemsCount = o.Items.Count }),
                TotalCount = total, Page = page, PageSize = pageSize
            };
            return Ok(dto);
        }

        /// <summary>Creates a new order with optional items.</summary>
        [HttpPost]
        [ProducesResponseType(typeof(OrderReadDto),
StatusCodes.Status201Created)]
        public async Task<IActionResult> Create([FromBody] OrderCreateDto body)
        {
            var uid = HttpContext.RequireUserId();
            var orderNo = $"ORD-{DateTime.UtcNow:yyyyMMddHHmmss}-
{Guid.NewGuid().ToString("N")[..6]}";
            var orderId = await _orders.CreateOrderAsync(uid, orderNo);

            foreach (var it in body.Items)
            {
                var p = await _products.GetByIdAsync(uid, it.ProductId);
                var price = (it.Price.HasValue && it.Price.Value > 0) ?
it.Price.Value : p.Price;
                await _orders.AddItemAsync(uid, orderId, it.ProductId, it.Qty,
price);
            }

            var order = await _orders.GetAsync(uid, orderId);
            var dto = new OrderReadDto
            {
                Id = order.Id, OrderNo = order.OrderNo, CreatedAt =
order.CreatedAt, Total = order.Total,
                Items = order.Items.Select(i => new OrderItemReadDto
                { Id = i.Id, ProductId = i.ProductId, ProductName =
i.ProductName, Qty = i.Qty, Price = i.Price }).ToList()
            };
            return CreatedAtAction(nameof(Get), new { id = order.Id }, dto);
        }

        /// <summary>Gets an order by id (with items).</summary>
        [HttpGet("{id:int}")]
        [ProducesResponseType(typeof(OrderReadDto), StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Get(int id)
        {
            var uid = HttpContext.RequireUserId();
            var order = await _orders.GetAsync(uid, id);
            return Ok(new OrderReadDto
            {
```

```
                Id = order.Id, OrderNo = order.OrderNo, CreatedAt =
order.CreatedAt, Total = order.Total,
                Items = order.Items.Select(i => new OrderItemReadDto
                { Id = i.Id, ProductId = i.ProductId, ProductName =
i.ProductName, Qty = i.Qty, Price = i.Price }).ToList()
            });
        }

        /// <summary>Adds a single item to an order.</summary>
        [HttpPost("{id:int}/items")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        public async Task<IActionResult> AddItem(int id, [FromBody]
OrderCreateItemDto body)
        {
            var uid = HttpContext.RequireUserId();
            var p = await _products.GetByIdAsync(uid, body.ProductId);
            var price = (body.Price.HasValue && body.Price.Value > 0) ?
body.Price.Value : p.Price;
            await _orders.AddItemAsync(uid, id, body.ProductId, body.Qty,
price);
            return NoContent();
        }

        /// <summary>Deletes an order.</summary>
        [HttpDelete("{id:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> Delete(int id)
        {
            var uid = HttpContext.RequireUserId();
            await _orders.DeleteAsync(uid, id);
            return NoContent();
        }

        /// <summary>Deletes an order item.</summary>
        [HttpDelete("{orderId:int}/items/{itemId:int}")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public async Task<IActionResult> DeleteItem(int orderId, int itemId)
        {
            var uid = HttpContext.RequireUserId();
            await _orders.DeleteItemAsync(uid, orderId, itemId);
            return NoContent();
        }
    }
}
```

Your repositories, models, DTOs, middleware, and logging from the prior code remain the same (remove duplicate models; keep one copy each).

Frontend (Razor) using `fetch()` to call APIs

Views/Shared/_Layout.cshtml

```
@using Microsoft.AspNetCore.Http
@{ var user = Context?.Session?.GetString("auth_user"); }
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>EmployeeCrudPdf</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/
bootstrap.min.css" rel="stylesheet" />
  <link href="/css/site.css" rel="stylesheet" />
</head>
<body class="bg-light">
<nav class="navbar navbar-expand navbar-dark bg-dark mb-3">
  <div class="container">
    <a class="navbar-brand" asp-controller="ApiFront" asp-action="Index">Home</
a>
    <a class="nav-link text-white" asp-controller="ApiFront" asp-
action="Employees">Employees</a>
    <a class="nav-link text-white" asp-controller="ApiFront" asp-
action="Products">Products</a>
    <a class="nav-link text-white" asp-controller="ApiFront" asp-
action="Orders">Orders</a>
    <div class="ms-auto d-flex align-items-center gap-2">
      @if (!string.IsNullOrEmpty(user))
      {
        <span class="text-white-50 me-2">Hello, @user</span>
        <button class="btn btn-outline-light btn-sm" id="btnLogout">Logout</
button>
      }
      else
      {
        <a class="btn btn-outline-light btn-sm" asp-controller="AccountMvc" asp-
action="Login">Login Page</a>
      }
      <a class="nav-link text-white" href="/swagger">Swagger</a>
    </div>
  </div>
```

```
</nav>
<div class="container">
  @RenderBody()
</div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/
jquery.min.js"></script>
<script src="/js/api-helpers.js"></script>
<script>
  document.getElementById('btnLogout')?.addEventListener('click', async () => {
    const res = await fetch('/api/auth/logout', { method: 'POST' });
    if (res.status === 204) location.reload(); else alert('Logout failed');
  });
</script>
@RenderSection("Scripts", required: false)
</body>
</html>
```

Controllers/ApiFrontController.cs (simple router to Razor pages)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EmployeeCrudPdf.Controllers
{
    public class ApiFrontController : Controller
    {
        public IActionResult Index() => View();
        [Authorize] public IActionResult Employees() => View();
        [Authorize] public IActionResult Products() => View();
        [Authorize] public IActionResult Orders() => View();
    }
}
```

Views/ApiFront/Index.cshtml

```
@{ ViewData["Title"] = "Home"; }
<div class="p-4 rounded bg-white shadow-sm">
  <h3>API-Driven Pages</h3>
  <p>Use these links to interact with the backend via REST APIs:</p>
  <ul>
    <li><a asp-action="Employees">Employees (fetch + pagination + search)</a></
li>
    <li><a asp-action="Products">Products (fetch + pagination + search)</a></li>
    <li><a asp-action="Orders">Orders (create/list/items)</a></li>
  </ul>
```

```
  <p class="mb-0">Use <a href="/swagger">Swagger</a> to test all endpoints,
including <code>/api/auth/*</code>.</p>
</div>
```

wwwroot/js/api-helpers.js

```
async function apiGet(url) {
  const res = await fetch(url, { method: 'GET' });
  if (!res.ok) throw new Error(await res.text());
  return await res.json();
}
async function apiSend(url, method, body) {
  const res = await fetch(url, {
    method,
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(body)
  });
  if (res.status === 204) return null;
  if (res.status === 201 || res.status === 200) return await res.json();
  throw new Error(await res.text());
}
function renderPager(container, total, page, pageSize, onJump) {
  const totalPages = Math.max(1, Math.ceil(total / pageSize));
  container.innerHTML = '';
  for (let i = 1; i <= totalPages; i++) {
    const btn = document.createElement('button');
    btn.className = 'btn btn-sm ' + (i === page ? 'btn-primary me-1' : 'btn-
outline-primary me-1');
    btn.textContent = i;
    btn.onclick = () => onJump(i);
    container.appendChild(btn);
  }
}
window.Api = { apiGet, apiSend, renderPager };
```

Views/ApiFront/Employees.cshtml

```
@{ ViewData["Title"] = "Employees"; }
<div class="p-3 rounded bg-white shadow-sm">
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h4 class="mb-0">Employees</h4>
    <div class="input-group" style="max-width:380px;">
      <input id="emp_kw" class="form-control" placeholder="Search name/
department/email" />
      <button class="btn btn-outline-secondary" id="emp_search">Search</button>
```

```
    </div>
  </div>

  <form id="emp_create" class="row g-2 mb-3">
    <div class="col-md-3"><input required id="emp_name" class="form-control"
placeholder="Name" /></div>
    <div class="col-md-3"><input required id="emp_dept" class="form-control"
placeholder="Department" /></div>
    <div class="col-md-3"><input required id="emp_email" type="email"
class="form-control" placeholder="Email" /></div>
    <div class="col-md-2"><input required id="emp_salary" type="number"
step="0.01" class="form-control" placeholder="Salary" /></div>
    <div class="col-md-1 d-grid"><button class="btn btn-primary"
type="submit">Add</button></div>
  </form>

  <table class="table table-striped table-hover" id="emp_tbl">
    <thead><tr><th>Id</th><th>Name</th><th>Dept</th><th>Email</th><th
class="text-end">Salary</th><th></th></tr></thead>
    <tbody></tbody>
  </table>
  <div id="emp_pager" class="mt-2"></div>
</div>

@section Scripts {
<script>
  const state = { page: 1, pageSize: 10, q: '' };
  async function load() {
    const data = await Api.apiGet(`/api/employees?q=$
{encodeURIComponent(state.q)}&page=${state.page}&pageSize=${state.pageSize}`);
    const tb = document.querySelector('#emp_tbl tbody');
    tb.innerHTML = '';
    for (const e of data.items) {
      const tr = document.createElement('tr');
      tr.innerHTML = `
        <td>${e.id}</td>
        <td><input class="form-control form-control-sm" value="${e.name}" data-
id="${e.id}" data-f="name"/></td>
        <td><input class="form-control form-control-sm" value="${e.department}"
data-id="${e.id}" data-f="department"/></td>
        <td><input class="form-control form-control-sm" value="${e.email}" data-
id="${e.id}" data-f="email"/></td>
        <td class="text-end"><input class="form-control form-control-sm text-
end" value="${e.salary}" data-id="${e.id}" data-f="salary"/></td>
        <td class="text-end">
          <button class="btn btn-sm btn-outline-primary" onclick="saveEmp($
{e.id})">Save</button>
          <button class="btn btn-sm btn-outline-danger" onclick="delEmp($
```

```
{e.id})">Delete</button>
        </td>`;
      tb.appendChild(tr);
    }
    Api.renderPager(document.getElementById('emp_pager'), data.totalCount,
data.page, data.pageSize, p => { state.page = p; load(); });
  }
  async function saveEmp(id) {
    const row = [...document.querySelectorAll(`[data-id='${id}']`)];
    const body = {
      name: row.find(x=>x.dataset.f==='name').value,
      department: row.find(x=>x.dataset.f==='department').value,
      email: row.find(x=>x.dataset.f==='email').value,
      salary: parseFloat(row.find(x=>x.dataset.f==='salary').value)
    };
    await Api.apiSend(`/api/employees/${id}`, 'PUT', body);
    await load();
  }
  async function delEmp(id) {
    if (!confirm('Delete employee?')) return;
    await fetch(`/api/employees/${id}`, { method: 'DELETE' });
    await load();
  }
  document.getElementById('emp_search').onclick = () => { state.q =
document.getElementById('emp_kw').value; state.page = 1; load(); };
  document.getElementById('emp_create').addEventListener('submit', async (e) =>
{
    e.preventDefault();
    const body = {
      name: document.getElementById('emp_name').value,
      department: document.getElementById('emp_dept').value,
      email: document.getElementById('emp_email').value,
      salary: parseFloat(document.getElementById('emp_salary').value)
    };
    await Api.apiSend('/api/employees', 'POST', body);
    e.target.reset();
    await load();
  });
  document.addEventListener('DOMContentLoaded', load);
</script>
}
```

Views/ApiFront/Products.cshtml

```
@{ ViewData["Title"] = "Products"; }
<div class="p-3 rounded bg-white shadow-sm">
```

```
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h4 class="mb-0">Products</h4>
    <div class="input-group" style="max-width:380px;">
      <input id="p_kw" class="form-control" placeholder="Search name/category" /
>
      <button class="btn btn-outline-secondary" id="p_search">Search</button>
    </div>
  </div>
  <form id="p_create" class="row g-2 mb-3">
    <div class="col-md-5"><input required id="p_name" class="form-control"
placeholder="Name" /></div>
    <div class="col-md-3"><input required id="p_price" type="number" step="0.01"
class="form-control" placeholder="Price" /></div>
    <div class="col-md-3"><input id="p_category" class="form-control"
placeholder="Category" /></div>
    <div class="col-md-1 d-grid"><button class="btn btn-primary"
type="submit">Add</button></div>
  </form>
  <table class="table table-striped table-hover" id="p_tbl">
    <thead><tr><th>Id</th><th>Name</th><th class="text-end">Price</
th><th>Category</th><th></th></tr></thead>
    <tbody></tbody>
  </table>
  <div id="p_pager" class="mt-2"></div>
</div>
@section Scripts {
<script>
  const pst = { page: 1, pageSize: 10, q: '' };
  async function loadP() {
    const data = await Api.apiGet(`/api/products?q=${encodeURIComponent(pst.q)}
&page=${pst.page}&pageSize=${pst.pageSize}`);
    const tb = document.querySelector('#p_tbl tbody');
    tb.innerHTML = '';
    for (const p of data.items) {
      const tr = document.createElement('tr');
      tr.innerHTML = `
        <td>${p.id}</td>
        <td><input class="form-control form-control-sm" value="${p.name}" data-
id="${p.id}" data-f="name"/></td>
        <td class="text-end"><input class="form-control form-control-sm text-
end" value="${p.price}" data-id="${p.id}" data-f="price"/></td>
        <td><input class="form-control form-control-sm" value="${p.category ??
''}" data-id="${p.id}" data-f="category"/></td>
        <td class="text-end">
          <button class="btn btn-sm btn-outline-primary" onclick="saveP($
{p.id})">Save</button>
          <button class="btn btn-sm btn-outline-danger" onclick="delP($
{p.id})">Delete</button>
```

```
        </td>`;
      tb.appendChild(tr);
    }
    Api.renderPager(document.getElementById('p_pager'), data.totalCount,
data.page, data.pageSize, p => { pst.page = p; loadP(); });
  }
  async function saveP(id) {
    const row = [...document.querySelectorAll(`[data-id='${id}']`)];
    const body = {
      name: row.find(x=>x.dataset.f==='name').value,
      price: parseFloat(row.find(x=>x.dataset.f==='price').value),
      category: row.find(x=>x.dataset.f==='category').value || null
    };
    await Api.apiSend(`/api/products/${id}`, 'PUT', body);
    await loadP();
  }
  async function delP(id) {
    if (!confirm('Delete product?')) return;
    await fetch(`/api/products/${id}`, { method: 'DELETE' });
    await loadP();
  }
  document.getElementById('p_search').onclick = () => { pst.q =
document.getElementById('p_kw').value; pst.page = 1; loadP(); };
  document.getElementById('p_create').addEventListener('submit', async (e) => {
    e.preventDefault();
    const body = {
      name: document.getElementById('p_name').value,
      price: parseFloat(document.getElementById('p_price').value),
      category: document.getElementById('p_category').value || null
    };
    await Api.apiSend('/api/products', 'POST', body);
    e.target.reset();
    await loadP();
  });
  document.addEventListener('DOMContentLoaded', loadP);
</script>
}
```

Views/ApiFront/Orders.cshtml

```
@{ ViewData["Title"] = "Orders"; }
<div class="p-3 rounded bg-white shadow-sm">
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h4 class="mb-0">Orders</h4>
    <div class="input-group" style="max-width:380px;">
      <input id="o_kw" class="form-control" placeholder="Search order no" />
```

```
      <button class="btn btn-outline-secondary" id="o_search">Search</button>
    </div>
  </div>

  <form id="o_create" class="row g-2 mb-3">
    <div class="col-12"><strong>Create New Order (optional initial item)</
strong></div>
    <div class="col-md-3"><input id="oi_productId" type="number" class="form-
control" placeholder="Product Id" /></div>
    <div class="col-md-3"><input id="oi_qty" type="number" class="form-control"
placeholder="Qty" value="1" /></div>
    <div class="col-md-3"><input id="oi_price" type="number" step="0.01"
class="form-control" placeholder="Price (optional)" /></div>
    <div class="col-md-3 d-grid"><button class="btn btn-primary" type="submit">Create
Order</button></div>
  </form>

  <table class="table table-striped table-hover" id="o_tbl">
    <thead><tr><th>Id</th><th>Order No</th><th>Created</th><th class="text-
end">Items</th><th class="text-end">Total</th><th></th></tr></thead>
    <tbody></tbody>
  </table>
  <div id="o_pager" class="mt-2"></div>

  <div class="mt-4" id="o_details"></div>
</div>

@section Scripts {
<script>
  const ost = { page:1, pageSize:10, q:'' };
  async function loadO() {
    const data = await Api.apiGet(`/api/orders?q=${encodeURIComponent(ost.q)}
&page=${ost.page}&pageSize=${ost.pageSize}`);
    const tb = document.querySelector('#o_tbl tbody');
    tb.innerHTML = '';
    for (const o of data.items) {
      const tr = document.createElement('tr');
      tr.innerHTML = `
        <td>${o.id}</td>
        <td>${o.orderNo}</td>
        <td>${new Date(o.createdAt).toLocaleString()}</td>
        <td class="text-end">${o.itemsCount}</td>
        <td class="text-end">${o.total.toFixed(2)}</td>
        <td class="text-end">
          <button class="btn btn-sm btn-outline-secondary" onclick="viewO($
{o.id})">View</button>
          <button class="btn btn-sm btn-outline-danger" onclick="delO($
{o.id})">Delete</button>
```

```
        </td>`;
      tb.appendChild(tr);
    }
    Api.renderPager(document.getElementById('o_pager'), data.totalCount,
data.page, data.pageSize, p => { ost.page = p; loadO(); });
  }
  async function viewO(id) {
    const o = await Api.apiGet(`/api/orders/${id}`);
    const el = document.getElementById('o_details');
    let rows = o.items.map(i => `
      <tr>
        <td>${i.id}</td><td>${i.productName ?? ''} (#${i.productId})</td>
        <td class="text-end">${i.qty}</td>
        <td class="text-end">${i.price.toFixed(2)}</td>
        <td class="text-end">${(i.qty*i.price).toFixed(2)}</td>
        <td class="text-end"><button class="btn btn-sm btn-outline-danger"
onclick="delItem(${o.id},${i.id})">Delete</button></td>
      </tr>`).join('');
    el.innerHTML = `
      <div class="card">
        <div class="card-body">
          <div class="d-flex justify-content-between align-items-center mb-2">
            <strong>Order ${o.orderNo}</strong>
            <span>Total: ${o.total.toFixed(2)}</span>
          </div>
          <div class="row g-2 mb-2">
            <div class="col-md-3"><input id="ai_productId" type="number"
class="form-control" placeholder="Product Id" /></div>
            <div class="col-md-3"><input id="ai_qty" type="number" class="form-
control" placeholder="Qty" value="1" /></div>
            <div class="col-md-3"><input id="ai_price" type="number" step="0.01"
class="form-control" placeholder="Price (optional)" /></div>
            <div class="col-md-3 d-grid"><button class="btn btn-outline-primary"
onclick="addItem(${o.id})">Add Item</button></div>
          </div>
          <table class="table table-sm table-striped">
            <thead><tr><th>#</th><th>Product</th><th class="text-end">Qty</
th><th class="text-end">Price</th><th class="text-end">Line</th><th></th></tr></
thead>
            <tbody>${rows}</tbody>
          </table>
        </div>
      </div>`;
  }
  async function addItem(orderId) {
    const body = {
      productId: parseInt(document.getElementById('ai_productId').value),
      qty: parseInt(document.getElementById('ai_qty').value),
```

```
      price: parseFloat(document.getElementById('ai_price').value) || null
    };
    await Api.apiSend(`/api/orders/${orderId}/items`, 'POST', body);
    await viewO(orderId);
    await loadO();
  }
  async function delItem(orderId,itemId) {
    await fetch(`/api/orders/${orderId}/items/${itemId}`, { method: 'DELETE' });
    await viewO(orderId);
    await loadO();
  }
  async function delO(id) {
    if (!confirm('Delete order?')) return;
    await fetch(`/api/orders/${id}`, { method: 'DELETE' });
    await loadO();
    document.getElementById('o_details').innerHTML = '';
  }
  document.getElementById('o_search').onclick = () => { ost.q =
document.getElementById('o_kw').value; ost.page = 1; loadO(); };
  document.getElementById('o_create').addEventListener('submit', async (e) => {
    e.preventDefault();
    const pid = parseInt(document.getElementById('oi_productId').value || '0');
    const qty = parseInt(document.getElementById('oi_qty').value || '0');
    const price = parseFloat(document.getElementById('oi_price').value || '0');
    const items = (pid>0 && qty>0) ? [{ productId: pid, qty, price: price>0?
price:null }] : [];
    const created = await Api.apiSend('/api/orders', 'POST', { items });
    await loadO();
    await viewO(created.id);
    e.target.reset();
  });
  document.addEventListener('DOMContentLoaded', loadO);
</script>
}
```

SQL – schema (SSMS)

```
CREATE TABLE dbo.Users (
  Id INT IDENTITY(1,1) PRIMARY KEY,
  Username NVARCHAR(80) NOT NULL UNIQUE,
  Email NVARCHAR(200) NOT NULL UNIQUE,
  PasswordHash NVARCHAR(200) NOT NULL,
  CreatedAt DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
);
```

```
CREATE TABLE dbo.employees (
  id INT IDENTITY(1,1) PRIMARY KEY,
  user_id INT NOT NULL,
  name NVARCHAR(100) NOT NULL,
  department NVARCHAR(100) NOT NULL,
  email NVARCHAR(255) NOT NULL,
  salary DECIMAL(18,2) NOT NULL DEFAULT(0),
  CONSTRAINT FK_employees_users FOREIGN KEY (user_id) REFERENCES dbo.Users(Id),
  CONSTRAINT UQ_employees_user_email UNIQUE (user_id, email)
);

CREATE TABLE dbo.products (
  id INT IDENTITY(1,1) PRIMARY KEY,
  user_id INT NOT NULL,
  name NVARCHAR(120) NOT NULL,
  price DECIMAL(18,2) NOT NULL,
  category NVARCHAR(200) NULL,
  CONSTRAINT FK_products_users FOREIGN KEY (user_id) REFERENCES dbo.Users(Id),
  CONSTRAINT UQ_products_user_name UNIQUE (user_id, name)
);

CREATE TABLE dbo.orders (
  id INT IDENTITY(1,1) PRIMARY KEY,
  user_id INT NOT NULL,
  order_no NVARCHAR(32) NOT NULL,
  created_at DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
  CONSTRAINT FK_orders_users FOREIGN KEY (user_id) REFERENCES dbo.Users(Id),
  CONSTRAINT UQ_orders_user_orderno UNIQUE (user_id, order_no)
);

CREATE TABLE dbo.order_items (
  id INT IDENTITY(1,1) PRIMARY KEY,
  order_id INT NOT NULL,
  product_id INT NOT NULL,
  qty INT NOT NULL CHECK (qty > 0),
  price DECIMAL(18,2) NOT NULL CHECK (price >= 0),
  user_id INT NOT NULL,
  CONSTRAINT FK_order_items_orders FOREIGN KEY (order_id) REFERENCES
dbo.orders(id),
  CONSTRAINT FK_order_items_products FOREIGN KEY (product_id) REFERENCES
dbo.products(id)
);
```

README.md

```
# EmployeeCrudPdf – Full Stack (Auth + Employees + Products + Orders)

**Tech:** .NET 8, MVC + Web API, JWT (HttpOnly cookie), Dapper + SQL Server,
Swagger/OpenAPI, Razor + fetch(), Bootstrap 5.

## Run
1. Configure `appsettings.json` connection string.
2. Create tables in SQL Server (see schema in docs above).
3. `dotnet restore && dotnet run`.
4. Go to `https://localhost:xxxx/swagger` to test the API.
5. App homepage (API-driven pages): `https://localhost:xxxx/ApiFront`.

## Auth
- `POST /api/auth/signup` – Body `{ username, email, password }` → `204`, sets
`access_token` cookie.
- `POST /api/auth/login` – Body `{ usernameOrEmail, password }` → `204`, sets
cookie.
- `POST /api/auth/logout` – `204`, clears cookie.

## Entities
- Employees: `/api/employees` (search `q`, `page`, `pageSize`)
- Products: `/api/products` (search `q`, `page`, `pageSize`)
- Orders: `/api/orders` (create with `items[]`, list, get, add item, delete)

## Frontend
Razor pages under `Views/ApiFront` call the APIs with `fetch()`.
The JWT cookie is **HttpOnly** and auto-sent on same-origin requests.

## Validation
Server-side via DataAnnotations (400 on failure). Client-side via jQuery
Validate (optional for the Razor forms you keep).

## Notes
- Use HTTPS in prod and set cookie `Secure=true`.
- Remove duplicate model classes; fix `[Authorize]` typo in
`ProductsController`.
```

Integration Notes

- After **signup/login** (via Swagger or UI) the HttpOnly cookie will be set; then the API-driven pages (Employees/Products/Orders) will work because the cookie is sent automatically on `fetch`.

- If you host frontend separately (different origin), enable **CORS** and use `credentials: 'include'` on fetch.
- Swagger supports both **cookieAuth** and **Bearer** so you can test either way.

That’s the full start-to-end setup: backend + Swagger + Razor pages wired via `fetch()` to the REST APIs.