Awesome — here’s a **starter repo blueprint you can paste + build immediately** (Clean Architecture + Identity/JWT + org scoping + Docker Postgres). I’m optimizing for: **hireable** + **usable for your consulting portal**.

1) Repo structure (Clean Architecture)

```
ConsultingAuditPortal/
  src/
    CAP.Api/
      Controllers/
      Middleware/
      Program.cs
      appsettings.json
      appsettings.Development.json
    CAP.Application/
      Common/
      Auth/
      Audits/
      Orgs/
      Intakes/
    CAP.Domain/
      Entities/
      Enums/
      Common/
    CAP.Infrastructure/
      Auth/
      Data/
      Storage/
  tests/
    CAP.Tests.Unit/
    CAP.Tests.Integration/
  docker-compose.yml
  Directory.Build.props
  .github/workflows/ci.yml
  README.md
```

2) Create the solution + projects

Run from repo root:

```
dotnet new sln -n ConsultingAuditPortal

dotnet new webapi -n CAP.Api -o src/CAP.Api
dotnet new classlib -n CAP.Domain -o src/CAP.Domain
dotnet new classlib -n CAP.Application -o src/CAP.Application
dotnet new classlib -n CAP.Infrastructure -o src/CAP.Infrastructure

dotnet sln add src/CAP.Api/CAP.Api.csproj
dotnet sln add src/CAP.Domain/CAP.Domain.csproj
dotnet sln add src/CAP.Application/CAP.Application.csproj
dotnet sln add src/CAP.Infrastructure/CAP.Infrastructure.csproj

dotnet add src/CAP.Api/CAP.Api.csproj reference src/CAP.Application/CAP.Application.csproj
dotnet add src/CAP.Application/CAP.Application.csproj reference
src/CAP.Domain/CAP.Domain.csproj
dotnet add src/CAP.Infrastructure/CAP.Infrastructure.csproj reference
src/CAP.Application/CAP.Application.csproj
dotnet add src/CAP.Infrastructure/CAP.Infrastructure.csproj reference
src/CAP.Domain/CAP.Domain.csproj
dotnet add src/CAP.Api/CAP.Api.csproj reference
src/CAP.Infrastructure/CAP.Infrastructure.csproj
```

3) NuGet packages (core)

```
# Infrastructure: EF Core + Postgres + Identity
dotnet add src/CAP.Infrastructure package Microsoft.EntityFrameworkCore
dotnet add src/CAP.Infrastructure package Microsoft.EntityFrameworkCore.Design
dotnet add src/CAP.Infrastructure package Npgsql.EntityFrameworkCore.PostgreSQL
dotnet add src/CAP.Infrastructure package Microsoft.AspNetCore.Identity.EntityFrameworkCore
dotnet add src/CAP.Infrastructure package
Microsoft.Extensions.Options.ConfigurationExtensions

# API: JWT auth + Swagger
dotnet add src/CAP.Api package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add src/CAP.Api package Swashbuckle.AspNetCore

# Optional but recommended: validation + logging
dotnet add src/CAP.Api package Serilog.AspNetCore
dotnet add src/CAP.Api package Serilog.Sinks.Console
dotnet add src/CAP.Api package FluentValidation.AspNetCore
```

4) Docker Postgres (local dev)

docker-compose.yml (repo root):

```
services:
  db:
    image: postgres:16
    container_name: cap_postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: cap
      POSTGRES_PASSWORD: cap_password
      POSTGRES_DB: cap_dev
    ports:
      - "5432:5432"
    volumes:
      - cap_pgdata:/var/lib/postgresql/data

volumes:
  cap_pgdata:
```

Run:

```
docker compose up -d
```

5) Domain models (minimum to be useful)

src/CAP.Domain/Entities/Organization.cs

```
namespace CAP.Domain.Entities;

public class Organization
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = default!;
    public DateTimeOffset CreatedAt { get; set; } = DateTimeOffset.UtcNow;

    public ICollection<OrganizationMember> Members { get; set; } = new
List<OrganizationMember>();
}
```

src/CAP.Domain/Entities/OrganizationMember.cs

```
namespace CAP.Domain.Entities;

public class OrganizationMember
{
    public Guid Id { get; set; } = Guid.NewGuid();

    public Guid OrganizationId { get; set; }
    public Organization Organization { get; set; } = default!;

    public string UserId { get; set; } = default!; // Identity user id (string)
    public string Role { get; set; } = "ClientViewer"; // Owner/Admin/ClientManager/ClientViewer

    public DateTimeOffset JoinedAt { get; set; } = DateTimeOffset.UtcNow;
}
```

src/CAP.Domain/Entities/Audit.cs

```
namespace CAP.Domain.Entities;

public class Audit
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public Guid OrganizationId { get; set; }
    public Organization Organization { get; set; } = default!;

    public string Title { get; set; } = default!;
    public string Status { get; set; } = "Draft"; // Draft/InReview/Delivered/InProgress/Closed

    public DateTimeOffset CreatedAt { get; set; } = DateTimeOffset.UtcNow;

    public ICollection<Finding> Findings { get; set; } = new List<Finding>();
}
```

src/CAP.Domain/Entities/Finding.cs

```
namespace CAP.Domain.Entities;

public class Finding
{
    public Guid Id { get; set; } = Guid.NewGuid();

    public Guid AuditId { get; set; }
    public Audit Audit { get; set; } = default!;

    public Guid OrganizationId { get; set; } // duplicated for fast scoping
    public string Category { get; set; } = "Ops";
    public string Severity { get; set; } = "Medium"; // Low/Medium/High
    public string Effort { get; set; } = "M"; // S/M/L

    public string Title { get; set; } = default!;
    public string Recommendation { get; set; } = default!;
    public string Status { get; set; } = "Identified"; // Identified/InProgress/Resolved

    public DateTimeOffset CreatedAt { get; set; } = DateTimeOffset.UtcNow;
}
```

6) Infrastructure: DbContext + Identity + global org scoping

src/CAP.Infrastructure/Auth/AppUser.cs

```
using Microsoft.AspNetCore.Identity;

namespace CAP.Infrastructure.Auth;

public class AppUser : IdentityUser
{
    // Keep minimal for now; add display name later if you want
}
```

src/CAP.Infrastructure/Data/AppDbContext.cs

```
using CAP.Domain.Entities;
using CAP.Infrastructure.Auth;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace CAP.Infrastructure.Data;

public class AppDbContext : IdentityDbContext<AppUser>
{
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) {}

    public DbSet<Organization> Organizations => Set<Organization>();
    public DbSet<OrganizationMember> OrganizationMembers => Set<OrganizationMember>();
    public DbSet<Audit> Audits => Set<Audit>();
    public DbSet<Finding> Findings => Set<Finding>();

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<Organization>(b =>
        {
            b.HasIndex(x => x.Name);
        });

        builder.Entity<OrganizationMember>(b =>
        {
            b.HasIndex(x => new { x.OrganizationId, x.UserId }).IsUnique();
        });

        builder.Entity<Audit>(b =>
        {
            b.HasIndex(x => x.OrganizationId);
        });

        builder.Entity<Finding>(b =>
        {
            b.HasIndex(x => x.OrganizationId);
            b.HasIndex(x => x.AuditId);
        });
    }
}
```

How org scoping will work (clean + interview-friendly)

- User authenticates via JWT
- JWT contains a **current org_id** claim (or header-based org selection later)
- Every query is filtered by that org_id

We'll implement this by:

1. extracting org_id from claims
2. passing it into a request-scoped service ICurrentOrg
3. controllers/repositories enforce OrganizationId == CurrentOrgId

src/CAP.Application/Common/ICurrentOrg.cs

```
namespace CAP.Application.Common;

public interface ICurrentOrg
{
    Guid OrganizationId { get; }
}
```

src/CAP.Api/Middleware/CurrentOrgFromClaims.cs

```
using System.Security.Claims;
using CAP.Application.Common;

namespace CAP.Api.Middleware;

public class CurrentOrgFromClaims : ICurrentOrg
{
    private readonly IHttpContextAccessor _http;

    public CurrentOrgFromClaims(IHttpContextAccessor http) => _http = http;

    public Guid OrganizationId
    {
        get
        {
            var user = _http.HttpContext?.User;
            var orgIdStr = user?.FindFirstValue("org_id");
            if (Guid.TryParse(orgIdStr, out var orgId)) return orgId;

            // For endpoints that don't require org yet (login/register)
            return Guid.Empty;
        }
    }
}
```

7) API config: appsettings + Program.cs (JWT + EF + Identity + Swagger)

src/CAP.Api/appsettings.Development.json

```
{
  "ConnectionStrings": {
    "Default":
"Host=localhost;Port=5432;Database=cap_dev;Username=cap;Password=cap_password"
  },
  "Jwt": {
    "Issuer": "CAP",
    "Audience": "CAP",
    "SigningKey": "DEV_ONLY_REPLACE_WITH_LONG_RANDOM_SECRET_32+_CHARS",
    "AccessTokenMinutes": 30
  }
}
```

src/CAP.Api/Program.cs

```
using System.Text;
using CAP.Api.Middleware;
using CAP.Application.Common;
using CAP.Infrastructure.Auth;
using CAP.Infrastructure.Data;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Serilog;

var builder = WebApplication.CreateBuilder(args);

builder.Host.UseSerilog((ctx, lc) => lc
    .ReadFrom.Configuration(ctx.Configuration)
    .WriteTo.Console());

builder.Services.AddHttpContextAccessor();

// Db
builder.Services.AddDbContext<AppDbContext>(opt =>
    opt.UseNpgsql(builder.Configuration.GetConnectionString("Default")));

// Identity
builder.Services
    .AddIdentityCore<AppUser>(opt =>
    {
        opt.Password.RequiredLength = 8;
        opt.User.RequireUniqueEmail = true;
    })
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<AppDbContext>();

// JWT
var jwtSection = builder.Configuration.GetSection("Jwt");
var signingKey = jwtSection["SigningKey"]!;
var keyBytes = Encoding.UTF8.GetBytes(signingKey);

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt =>
    {
        opt.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtSection["Issuer"],
            ValidAudience = jwtSection["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(keyBytes),
            ClockSkew = TimeSpan.FromSeconds(30)
        };
    });

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("OwnerOrAdmin", p =>
        p.RequireClaim("role", "Owner", "Admin"));
});

// Current Org
builder.Services.AddScoped<ICurrentOrg, CurrentOrgFromClaims>();

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSerilogRequestLogging();

app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

8) Auth endpoints (Register + Login) that issue org-scoped JWT

src/CAP.Api/Controllers/AuthController.cs

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using CAP.Domain.Entities;
using CAP.Infrastructure.Auth;
using CAP.Infrastructure.Data;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;

namespace CAP.Api.Controllers;

[ApiController]
[Route("api/auth")]
public class AuthController : ControllerBase
{
    private readonly UserManager<AppUser> _users;
    private readonly AppDbContext _db;
    private readonly IConfiguration _cfg;

    public AuthController(UserManager<AppUser> users, AppDbContext db, IConfiguration cfg)
    {
        _users = users;
        _db = db;
        _cfg = cfg;
    }

    public record RegisterRequest(string Email, string Password, string OrganizationName);
    public record LoginRequest(string Email, string Password);
    public record AuthResponse(string AccessToken, Guid OrganizationId);

    [HttpPost("register")]
    public async Task<ActionResult<AuthResponse>> Register(RegisterRequest req)
    {
        // Create user
        var user = new AppUser { UserName = req.Email, Email = req.Email };
        var create = await _users.CreateAsync(user, req.Password);
        if (!create.Succeeded) return BadRequest(create.Errors);

        // Create org + membership (Owner)
        var org = new Organization { Name = req.OrganizationName };
        _db.Organizations.Add(org);

        _db.OrganizationMembers.Add(new OrganizationMember
        {
            OrganizationId = org.Id,
            UserId = user.Id,
            Role = "Owner"
        });

        await _db.SaveChangesAsync();

        var token = CreateJwt(user.Id, org.Id, "Owner");
        return Ok(new AuthResponse(token, org.Id));
    }

    [HttpPost("login")]
    public async Task<ActionResult<AuthResponse>> Login(LoginRequest req)
    {
        var user = await _users.Users.FirstOrDefaultAsync(x => x.Email == req.Email);
        if (user is null) return Unauthorized();

        var ok = await _users.CheckPasswordAsync(user, req.Password);
        if (!ok) return Unauthorized();

        // Pick the user's first org for now (later: allow org selection)
        var membership = await _db.OrganizationMembers
            .Where(m => m.UserId == user.Id)
            .OrderBy(m => m.JoinedAt)
            .FirstOrDefaultAsync();

        if (membership is null) return Unauthorized("No organization membership.");

        var token = CreateJwt(user.Id, membership.OrganizationId, membership.Role);
        return Ok(new AuthResponse(token, membership.OrganizationId));
    }

    private string CreateJwt(string userId, Guid orgId, string role)
    {
        var jwt = _cfg.GetSection("Jwt");
        var issuer = jwt["Issuer"]!;
        var audience = jwt["Audience"]!;
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwt["SigningKey"]!));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var claims = new List<Claim>
        {
            new(JwtRegisteredClaimNames.Sub, userId),
            new("org_id", orgId.ToString()),
            new("role", role)
        };

        var minutes = int.Parse(jwt["AccessTokenMinutes"] ?? "30");

        var token = new JwtSecurityToken(
            issuer: issuer,
            audience: audience,
            claims: claims,
            expires: DateTime.UtcNow.AddMinutes(minutes),
            signingCredentials: creds);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

9) First “real” controller: Audits (org-scoped)

src/CAP.Api/Controllers/AuditsController.cs

```
using CAP.Application.Common;
using CAP.Domain.Entities;
using CAP.Infrastructure.Data;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace CAP.Api.Controllers;

[ApiController]
[Route("api/audits")]
[Authorize]
public class AuditsController : ControllerBase
{
    private readonly AppDbContext _db;
    private readonly ICurrentOrg _org;

    public AuditsController(AppDbContext db, ICurrentOrg org)
    {
        _db = db;
        _org = org;
    }

    public record CreateAuditRequest(string Title);

    [HttpGet]
    public async Task<ActionResult<List<Audit>>> List()
    {
        if (_org.OrganizationId == Guid.Empty) return Unauthorized("Missing org_id claim.");
        var items = await _db.Audits
            .Where(a => a.OrganizationId == _org.OrganizationId)
            .OrderByDescending(a => a.CreatedAt)
            .ToListAsync();

        return Ok(items);
    }

    [HttpPost]
    [Authorize(Policy = "OwnerOrAdmin")]
    public async Task<ActionResult<Audit>> Create(CreateAuditRequest req)
    {
        if (_org.OrganizationId == Guid.Empty) return Unauthorized("Missing org_id claim.");

        var audit = new Audit
        {
            OrganizationId = _org.OrganizationId,
            Title = req.Title,
            Status = "Draft"
        };

        _db.Audits.Add(audit);
        await _db.SaveChangesAsync();

        return Ok(audit);
    }
}
```

10) Migrations + run

From repo root:

```
dotnet tool install --global dotnet-ef
dotnet ef migrations add Initial --project src/CAP.Infrastructure --startup-project src/CAP.Api
dotnet ef database update --project src/CAP.Infrastructure --startup-project src/CAP.Api

dotnet run --project src/CAP.Api
```

Open Swagger:

- `https://localhost:<port>/swagger`

Flow:

1. `POST /api/auth/register`
2. Copy token → Swagger “Authorize” → `Bearer <token>`
3. `POST /api/audits` then `GET /api/audits`

11) GitHub Actions CI (build + test)

`.github/workflows/ci.yml`:

```
name: ci

on:
  push:
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-dotnet@v4
        with:
          dotnet-version: "8.0.x"
      - name: Restore
        run: dotnet restore
      - name: Build
        run: dotnet build -c Release --no-restore
      - name: Test
        run: dotnet test -c Release --no-build
```

What you have after this scaffold

- Real **multi-tenant** backbone (org_id claim + enforced scoping)
- Identity + JWT auth
- Clean Architecture shape
- Docker Postgres local dev
- A real “Audits” slice to expand into Findings/Intake/Uploads next