EF Core CRUD for ASP.NET Core Identity (`ApplicationDbContext`) – komplett kokebok

> Denne guiden viser **hvordan du gjør CRUD-operasjoner direkte mot Identity-tabellene** via `ApplicationDbContext : IdentityDbContext<IdentityUser, IdentityRole, string>`.
> Du får eksempler for **alle DbSets** Identity består av:
>
> - `Users` (`AspNetUsers`)
> - `Roles` (`AspNetRoles`)
> - `UserRoles` (`AspNetUserRoles`)
> - `UserClaims` (`AspNetUserClaims`)
> - `RoleClaims` (`AspNetRoleClaims`)
> - `UserLogins` (`AspNetUserLogins`)
> - `UserTokens` (`AspNetUserTokens`)
>
> ⚠ **Anbefaling:** I vanlig applikasjonskode bør du oftest bruke **`UserManager`**, **`RoleManager`** og **`SignInManager`**.
> Direkte bruk av `DbContext` er likevel nyttig for *seeding, adminscripts, migreringer og spesialspørringer*.

Oppsett og nyttige imports

```
using System.Security.Claims;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

// Din DbContext
public class ApplicationDbContext : IdentityDbContext<IdentityUser, IdentityRole,
string>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options) { }
}
```

DI/bruk (eksempel i en service/controller):

```
public class IdentityCrudService
{
    private readonly ApplicationDbContext _db;
    public IdentityCrudService(ApplicationDbContext db) => _db = db;

    // Bruk _db i async-metoder under
}
```

Normalize-hjelp (Identity forventer normalt UPPERCASE):

```
static string N(string v) => v.ToUpperInvariant();
```

---

1) Users (`AspNetUsers`)

Create

```
public async Task<string> CreateUserAsync(string email, string password)
{
    var user = new IdentityUser
    {
        Id = Guid.NewGuid().ToString(),
        UserName = email,
        NormalizedUserName = N(email),
        Email = email,
        NormalizedEmail = N(email),
        EmailConfirmed = true,              // sett etter behov
        SecurityStamp = Guid.NewGuid().ToString(),
        ConcurrencyStamp = Guid.NewGuid().ToString(),
        PhoneNumberConfirmed = false,
        TwoFactorEnabled = false,
        LockoutEnabled = true,
        AccessFailedCount = 0
    };

    // Passordhash (hvis du ikke bruker UserManager)
    var hasher = new PasswordHasher<IdentityUser>();
    user.PasswordHash = hasher.HashPassword(user, password);

    _db.Users.Add(user);
    await _db.SaveChangesAsync();
    return user.Id;
}
```

Read

```
public Task<IdentityUser?> FindUserByEmailAsync(string email) =>
    _db.Users.AsNoTracking().FirstOrDefaultAsync(u => u.NormalizedEmail ==
N(email));
```

Update

```
public async Task UpdateUserEmailAsync(string userId, string newEmail)
{
    var user = await _db.Users.FirstAsync(u => u.Id == userId);
    user.Email = newEmail;
    user.NormalizedEmail = N(newEmail);
    user.UserName = newEmail;
    user.NormalizedUserName = N(newEmail);
    user.ConcurrencyStamp = Guid.NewGuid().ToString(); // god vane
    await _db.SaveChangesAsync();
}
```

Delete

```
public async Task DeleteUserAsync(string userId)
{
    var user = await _db.Users.FirstAsync(u => u.Id == userId);
    _db.Users.Remove(user);
    await _db.SaveChangesAsync();
}
```

2) Roles (`AspNetRoles`)

Create

```
public async Task<string> CreateRoleAsync(string roleName)
{
    var role = new IdentityRole
    {
        Id = Guid.NewGuid().ToString(),
        Name = roleName,
        NormalizedName = N(roleName),
        ConcurrencyStamp = Guid.NewGuid().ToString()
    };
    _db.Roles.Add(role);
    await _db.SaveChangesAsync();
    return role.Id;
}
```

Read

```
public Task<IdentityRole?> FindRoleAsync(string roleName) =>
    _db.Roles.AsNoTracking().FirstOrDefaultAsync(r => r.NormalizedName ==
N(roleName));
```

Update

```
public async Task RenameRoleAsync(string roleId, string newName)
{
    var role = await _db.Roles.FirstAsync(r => r.Id == roleId);
    role.Name = newName;
    role.NormalizedName = N(newName);
    role.ConcurrencyStamp = Guid.NewGuid().ToString();
    await _db.SaveChangesAsync();
}
```

Delete

```
public async Task DeleteRoleAsync(string roleId)
{
    var role = await _db.Roles.FirstAsync(r => r.Id == roleId);
    _db.Roles.Remove(role);
    await _db.SaveChangesAsync();
}
```

3) UserRoles (`AspNetUserRoles`) — kobling bruker↔rolle

Add user to role

```
public async Task AddUserToRoleAsync(string userId, string roleName)
{
    var role = await _db.Roles.FirstAsync(r => r.NormalizedName == N(roleName));

    var link = new IdentityUserRole<string> { UserId = userId, RoleId = role.Id };
    // Unngå dubletter
    var exists = await _db.UserRoles.AnyAsync(ur => ur.UserId == userId &&
ur.RoleId == role.Id);
    if (!exists)
    {
        _db.UserRoles.Add(link);
        await _db.SaveChangesAsync();
    }
}
```

Get user roles

```
public async Task<string[]> GetUserRolesAsync(string userId)
{
    var q =
        from ur in _db.UserRoles
        join r in _db.Roles on ur.RoleId equals r.Id
        where ur.UserId == userId
        select r.Name!;
    return await q.ToArrayAsync();
}
```

Remove user from role

```
public async Task RemoveUserFromRoleAsync(string userId, string roleName)
{
    var roleId = await _db.Roles
        .Where(r => r.NormalizedName == N(roleName))
        .Select(r => r.Id)
        .FirstAsync();

    var link = await _db.UserRoles.FirstOrDefaultAsync(ur => ur.UserId == userId
&& ur.RoleId == roleId);
    if (link != null)
    {
        _db.UserRoles.Remove(link);
        await _db.SaveChangesAsync();
    }
}
```

4) UserClaims (`AspNetUserClaims`)

Add claim to user

```
public async Task<int> AddUserClaimAsync(string userId, string type, string value)
{
    var claim = new IdentityUserClaim<string>
    {
        UserId = userId,
        ClaimType = type,
        ClaimValue = value
    };
    _db.UserClaims.Add(claim);
    await _db.SaveChangesAsync();
    return claim.Id; // PK (int)
}
```

List user claims

```
public Task<IdentityUserClaim<string>[]> GetUserClaimsAsync(string userId) =>
    _db.UserClaims.AsNoTracking().Where(c => c.UserId == userId).ToArrayAsync();
```

Update claim

```
public async Task UpdateUserClaimAsync(int claimId, string newValue)
{
    var c = await _db.UserClaims.FirstAsync(x => x.Id == claimId);
    c.ClaimValue = newValue;
    await _db.SaveChangesAsync();
}
```

Remove claim

```
public async Task RemoveUserClaimAsync(int claimId)
{
    var c = await _db.UserClaims.FirstAsync(x => x.Id == claimId);
    _db.UserClaims.Remove(c);
    await _db.SaveChangesAsync();
}
```

5) RoleClaims (`AspNetRoleClaims`)

Add claim to role

```
public async Task<int> AddRoleClaimAsync(string roleName, string type, string
value)
{
    var roleId = await _db.Roles.Where(r => r.NormalizedName ==
N(roleName)).Select(r => r.Id).FirstAsync();
    var rc = new IdentityRoleClaim<string>
    {
        RoleId = roleId,
        ClaimType = type,
        ClaimValue = value
    };
    _db.RoleClaims.Add(rc);
    await _db.SaveChangesAsync();
    return rc.Id;
}
```

List role claims

```
public Task<IdentityRoleClaim<string>[]> GetRoleClaimsAsync(string roleName) =>
    (from r in _db.Roles
     join rc in _db.RoleClaims on r.Id equals rc.RoleId
     where r.NormalizedName == N(roleName)
     select rc)
    .AsNoTracking()
    .ToArrayAsync();
```

Update & Delete

```
public async Task UpdateRoleClaimAsync(int roleClaimId, string newValue)
{
    var rc = await _db.RoleClaims.FirstAsync(x => x.Id == roleClaimId);
    rc.ClaimValue = newValue;
    await _db.SaveChangesAsync();
}

public async Task RemoveRoleClaimAsync(int roleClaimId)
{
    var rc = await _db.RoleClaims.FirstAsync(x => x.Id == roleClaimId);
    _db.RoleClaims.Remove(rc);
    await _db.SaveChangesAsync();
}
```

6) UserLogins (`AspNetUserLogins`) – eksterne logins (Google, MS, osv.)

> Primært håndtert av `SignInManager`/`UserManager` via eksterne providere. Direkte CRUD kan være nyttig for admin-operasjoner.

Add login

```
public async Task AddUserLoginAsync(string userId, string provider, string
providerKey, string? displayName = null)
{
    var login = new IdentityUserLogin<string>
    {
        LoginProvider = provider,       // f.eks. "Google"
        ProviderKey = providerKey,      // unik nøkkel fra provider
        ProviderDisplayName = displayName,
        UserId = userId
    };

    var exists = await _db.UserLogins.AnyAsync(l =>
        l.LoginProvider == provider && l.ProviderKey == providerKey && l.UserId ==
userId);

    if (!exists)
    {
        _db.UserLogins.Add(login);
        await _db.SaveChangesAsync();
    }
}
```

Find logins for user

```
public Task<IdentityUserLogin<string>[]> GetUserLoginsAsync(string userId) =>
    _db.UserLogins.AsNoTracking().Where(l => l.UserId == userId).ToArrayAsync();
```

Remove login

```
public async Task RemoveUserLoginAsync(string userId, string provider, string
providerKey)
{
    var login = await _db.UserLogins
        .FirstOrDefaultAsync(l => l.UserId == userId && l.LoginProvider ==
provider && l.ProviderKey == providerKey);
    if (login != null)
    {
        _db.UserLogins.Remove(login);
        await _db.SaveChangesAsync();
    }
}
```

7) UserTokens (`AspNetUserTokens`) – tokens per bruker

> Brukes av Identity for f.eks. authenticator keys, reset-passord, etc.

Add/update token

```
public async Task UpsertUserTokenAsync(string userId, string provider, string
name, string value)
{
    var token = await _db.UserTokens
        .FirstOrDefaultAsync(t => t.UserId == userId && t.LoginProvider ==
provider && t.Name == name);

    if (token is null)
    {
        token = new IdentityUserToken<string>
        {
            UserId = userId,
            LoginProvider = provider, // f.eks. "Default"
            Name = name,              // f.eks. "RefreshToken"
            Value = value
        };
        _db.UserTokens.Add(token);
    }
    else
    {
        token.Value = value;
        _db.UserTokens.Update(token);
    }
    await _db.SaveChangesAsync();
}
```

Read & delete

```
public Task<IdentityUserToken<string>?> GetUserTokenAsync(string userId, string
provider, string name) =>
    _db.UserTokens.AsNoTracking()
       .FirstOrDefaultAsync(t => t.UserId == userId && t.LoginProvider == provider
&& t.Name == name);

public async Task RemoveUserTokenAsync(string userId, string provider, string
name)
{
    var token = await _db.UserTokens
        .FirstOrDefaultAsync(t => t.UserId == userId && t.LoginProvider ==
provider && t.Name == name);
    if (token != null)
    {
        _db.UserTokens.Remove(token);
```

```
            await _db.SaveChangesAsync();
        }
    }
```

---

8) Nyttige spørringer og include-eksempler

Hent bruker med roller og claims

```
public async Task<object?> GetUserWithRolesAndClaimsAsync(string email)
{
    var user = await _db.Users.FirstOrDefaultAsync(u => u.NormalizedEmail ==
N(email));
    if (user is null) return null;

    var roles =
        from ur in _db.UserRoles
        join r in _db.Roles on ur.RoleId equals r.Id
        where ur.UserId == user.Id
        select r.Name!;

    var claims = _db.UserClaims.Where(c => c.UserId == user.Id);

    return new
    {
        user.Id,
        user.Email,
        Roles = await roles.ToArrayAsync(),
        Claims = await claims.Select(c => new { c.ClaimType, c.ClaimValue
}).ToArrayAsync()
    };
}
```

Finn alle brukere i en rolle

```
public Task<string[]> GetUsersInRoleAsync(string roleName)
{
    var q =
        from r in _db.Roles
        join ur in _db.UserRoles on r.Id equals ur.RoleId
        join u in _db.Users on ur.UserId equals u.Id
        where r.NormalizedName == N(roleName)
        select u.Email!;
    return q.ToArrayAsync();
}
```

9) Tips & fallgruver

- **Bruk Managers i forretningslogikk** når mulig (de tar hensyn til politikk som lockout, validering, tokenproviders).
- **Normalized** felter må være UPPERCASE.
- Ved **passordendring** direkte via DbContext: husk å generere **ny hash** (helst via `UserManager`), og oppdater `SecurityStamp`/`ConcurrencyStamp`.
- Ved **sletting av bruker/rolle**, sørg for å rydde opp i relasjoner (UserRoles, Claims, Logins, Tokens) om DB-relasjoner ikke håndterer cascading.
- Bruk **`AsNoTracking()`** for rene lesespørringer for ytelse.
- Pakk batch-operasjoner i **transaksjon** hvis flere tabeller oppdateres samtidig.

---

10) Hvor i struktur?

I prosjekt med *by-feature* + *Infrastructure*:

```
/src
 ├─ /Features
 │   └─ /IdentityAdmin        // om du lager admin-endepunkter
 │       └─ IdentityCrudService.cs
 └─ /Infrastructure
     └─ /Persistence
         └─ ApplicationDbContext.cs
```

---

11) Bonus: JWT-kontekst

Når du fyller **UserClaims** og **UserRoles**, kan du ta dem inn i JWT som `ClaimTypes.Role` og egendefinerte claims, og bruke `[Authorize(Roles="...")]` eller policies.
