EntityFrameworkConcurrency01 - Follow me agenda

Concurrency - LostUpdate

Tilføj optimistic concurrency til modellen

```
public class Blog
{
    public int Id { get; set; }
    public string Url { get; set; }
    public List<Post> Posts { get; } = [];
    [Timestamp] public byte[] RowVersion { get; private set; } = [];
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public Blog Blog { get; set; }
    [Timestamp] public byte[] RowVersion { get; private set; } = [];
}
```

Opdater databasen

```
Add-Migration Concurrency
Update-Database
```

Opret API controller for Blog

- Stå på Controller folderen.
- Højre klik og vælg add Controller
- Vælg API
- Vælg API Controller with actions, using Entity Framework
- Tryk Add
- I Model class - vælg Blog
- I DbContext class - vælg BloggingContext
- Tryk Add

Følgende controller genereres nu:

```
using EntityFrameworkConcurrency01.Database;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace EntityFrameworkConcurrency01.Controllers;

[Route("api/[controller]")]
[ApiController]
public class BlogsController : ControllerBase
{
    private readonly BloggingContext _context;

    public BlogsController(BloggingContext context)
    {
        _context = context;
    }

    // GET: api/Blogs
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Blog>>> GetBlogs()
    {
        return await _context.Blogs.ToListAsync();
    }

    // GET: api/Blogs/5
    [HttpGet("{id}")]
    public async Task<ActionResult<Blog>> GetBlog(int id)
    {
        var blog = await _context.Blogs.FindAsync(id);

        if (blog == null) return NotFound();

        return blog;
    }

    // PUT: api/Blogs/5
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?
linkid=2123754
    [HttpPut("{id}")]
    public async Task<IActionResult> PutBlog(int id, Blog blog)
    {
        if (id != blog.Id) return BadRequest();

        _context.Entry(blog).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!BlogExists(id))
                return NotFound();
```

```
            throw;
        }

        return NoContent();
    }

    // POST: api/Blogs
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?
linkid=2123754
    [HttpPost]
    public async Task<ActionResult<Blog>> PostBlog(Blog blog)
    {
        _context.Blogs.Add(blog);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetBlog", new { id = blog.Id }, blog);
    }

    // DELETE: api/Blogs/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteBlog(int id)
    {
        var blog = await _context.Blogs.FindAsync(id);
        if (blog == null) return NotFound();

        _context.Blogs.Remove(blog);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool BlogExists(int id)
    {
        return _context.Blogs.Any(e => e.Id == id);
    }
}
```

Opret en Blog med Swagger

Brug Swagger til at oprette en Blog i databasen

Test concurrency fejl med Swagger

Brug Swagger til at fremprovokere en concurrency fejl

Korriger koden for "PutBlog"

```
    [HttpPut("{id}")]
    public async Task<IActionResult> PutBlog(int id, Blog updatedBlog)
    {
        if (id != updatedBlog.Id) return BadRequest();

        var blog = await _context.Blogs.FindAsync(id);
        if (blog == null) return NotFound();

        blog.Url = updatedBlog.Url;
        _context.Entry(blog).Property(p => p.RowVersion).OriginalValue =
updatedBlog.RowVersion;


        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException e)
        {
            return Conflict(e.Entries.Single().GetDatabaseValues());
        }

        return NoContent();
    }
```

Test concurrency fejl med Swagger

Brug Swagger til at fremprovokere en concurrency fejl

Concurrency - Phantom

Tilføj UnitOfWork klasser

```
using System.Data;
using EntityFrameworkConcurrency01.Database;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage;

namespace EntityFrameworkConcurrency01;

public interface IUnitOfWork
{
    void Commit();
    void Rollback();
    void BeginTransaction(IsolationLevel isolationLevel =
IsolationLevel.Serializable);
}

public class UnitOfWork : IUnitOfWork
```

```
{
    private readonly BloggingContext _db;
    private IDbContextTransaction? _transaction;

    public UnitOfWork(BloggingContext context)
    {
        _db = context;
    }

    void IUnitOfWork.BeginTransaction(IsolationLevel isolationLevel)
    {
        if (_db.Database.CurrentTransaction != null) return;
        _transaction = _db.Database.BeginTransaction(isolationLevel);
    }

    void IUnitOfWork.Commit()
    {
        if (_transaction == null) throw new Exception("You must call
'BeginTransaction' before Commit is called");
        _transaction.Commit();
        _transaction.Dispose();
    }

    void IUnitOfWork.Rollback()
    {
        if (_transaction == null) throw new Exception("You must call
'BeginTransaction' before Rollback is called");
        _transaction.Rollback();
        _transaction.Dispose();
    }
}
```

Tilføj i program.cs

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

```
using EntityFrameworkConcurrency01;
using EntityFrameworkConcurrency01.Database;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Add-Migration InitialCreate
// Update-Database
builder.Services.AddDbContext<BloggingContext>(opt =>
```

```
opt.UseSqlServer(builder.Configuration.GetConnectionString("SqlConnection")));

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();
```

Korriger koden for "PutBlog"

```
using EntityFrameworkConcurrency01.Database;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace EntityFrameworkConcurrency01.Controllers;

[Route("api/[controller]")]
[ApiController]
public class BlogsController : ControllerBase
{
    private readonly BloggingContext _context;
    private readonly IUnitOfWork _unitOfWork;

    public BlogsController(BloggingContext context, IUnitOfWork unitOfWork)
    {
        _context = context;
        _unitOfWork = unitOfWork;
    }

    // GET: api/Blogs
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Blog>>> GetBlogs()
    {
        return await _context.Blogs.ToListAsync();
    }

    // GET: api/Blogs/5
    [HttpGet("{id}")]
    public async Task<ActionResult<Blog>> GetBlog(int id)
    {
        var blog = await _context.Blogs.FindAsync(id);

        if (blog == null) return NotFound();
```

```
        return blog;
    }

    // PUT: api/Blogs/5
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?
linkid=2123754
    [HttpPut("{id}")]
    public async Task<IActionResult> PutBlog(int id, Blog updatedBlog)
    {
        if (id != updatedBlog.Id) return BadRequest();

        try
        {
            _unitOfWork.BeginTransaction();


            var blog = await _context.Blogs.FindAsync(id);
            if (blog == null) return NotFound();

            blog.Url = updatedBlog.Url;
            _context.Entry(blog).Property(p => p.RowVersion).OriginalValue =
updatedBlog.RowVersion;

            await _context.SaveChangesAsync();

            _unitOfWork.Commit();
        }
        catch (DbUpdateConcurrencyException e)
        {
            try
            {
                _unitOfWork.Rollback();
            }
            catch (Exception ex)
            {
                throw new Exception($"Rollback failed: {ex.Message}", e);
            }

            return Conflict(e.Entries.Single().GetDatabaseValues());
        }
        catch (Exception e)
        {
            try
            {
                _unitOfWork.Rollback();
            }
            catch (Exception ex)
            {
                return BadRequest($"Rollback failed: {ex.Message}");
            }

            throw;
        }

        return NoContent();
```

```
    }

    // POST: api/Blogs
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?
linkid=2123754
    [HttpPost]
    public async Task<ActionResult<Blog>> PostBlog(Blog blog)
    {
        _context.Blogs.Add(blog);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetBlog", new { id = blog.Id }, blog);
    }

    // DELETE: api/Blogs/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteBlog(int id)
    {
        var blog = await _context.Blogs.FindAsync(id);
        if (blog == null) return NotFound();

        _context.Blogs.Remove(blog);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool BlogExists(int id)
    {
        return _context.Blogs.Any(e => e.Id == id);
    }
}
```