**Mengimplementasikan Event-Driven Architecture di Aplikasi DompetKU**

Aplikasi DompetKU membutuhkan arsitektur yang fleksibel untuk memisahkan logika bisnis dari komponen utama. Event-Driven Architecture (EDA) menjadi solusi untuk memungkinkan komponen-komponen bereaksi terhadap event (misalnya: transaksi baru) tanpa ketergantungan langsung. Artikel ini menjelaskan implementasi EDA from scratch tanpa library eksternal, menggunakan C# murni.

**Konsep Dasar**

Event: Kejadian penting dalam sistem (contoh: transaksi baru dibuat).

Publisher: Komponen yang memicu event.

Subscriber: Komponen yang merespons event.

Decoupling: Pemisahan logika antar komponen.

**Langkah 1: Definisi Event**

Buat kelas untuk merepresentasikan event transaksi baru:

// Models/Events/TransactionCreatedEvent.cs

public class TransactionCreatedEvent

{

public Transaction Transaction { get; }

public TransactionCreatedEvent(Transaction transaction)

{

Transaction = transaction;

}

}

**Langkah 2: Implementasi Event Publisher**

Buat sistem publikasi/subscribsi event sederhana:

Interface untuk Publisher

csharp

// Interfaces/IEventPublisher.cs

public interface IEventPublisher

{

void Subscribe<TEvent>(Action<TEvent> handler);

void Publish<TEvent>(TEvent @event);

}

Implementasi Publisher

csharp

// Services/SimpleEventPublisher.cs

public class SimpleEventPublisher : IEventPublisher

{

private readonly Dictionary<Type, List<Delegate>> \_handlers = new();

public void Subscribe<TEvent>(Action<TEvent> handler)

{

var eventType = typeof(TEvent);

if (!\_handlers.ContainsKey(eventType))

{

\_handlers[eventType] = new List<Delegate>();

}

\_handlers[eventType].Add(handler);

}

public void Publish<TEvent>(TEvent @event)

{

var eventType = typeof(TEvent);

if (\_handlers.TryGetValue(eventType, out var handlers))

{

foreach (var handler in handlers)

{

((Action<TEvent>)handler)(@event);

}

}

}

}

Penjelasan Kode:

Subscribe: Menyimpan handler untuk tipe event tertentu.

Publish: Memanggil semua handler yang terkait dengan event.

**Langkah 3: Registrasi Service**

Tambahkan publisher ke dependency injection di Program.cs:

builder.Services.AddSingleton<IEventPublisher, SimpleEventPublisher>();

**Langkah 4: Implementasi Event Handler**

Contoh handler untuk update laporan:

// Services/ReportUpdater.cs

public class ReportUpdater

{

private readonly AppDbContext \_db;

public ReportUpdater(

IEventPublisher publisher,

AppDbContext db)

{

\_db = db;

// Subscribe ke event

publisher.Subscribe<TransactionCreatedEvent>(HandleTransactionCreated);

}

private void HandleTransactionCreated(TransactionCreatedEvent @event)

{

// Contoh: Update laporan

var transaction = @event.Transaction;

Console.WriteLine($"[EVENT] Transaksi baru: {transaction.Description}");

// Hitung total pendapatan/pengeluaran

var total = \_db.Transactions

.Where(t => t.Type == transaction.Type)

.Sum(t => t.Amount);

Console.WriteLine($"Total terbaru: {total:C}");

}

}

**Langkah 5: Modifikasi Controller**

Contoh: PendapatanController.cs

public class PendapatanController : Controller

{

private readonly AppDbContext \_db;

private readonly IEventPublisher \_eventPublisher;

public PendapatanController(

AppDbContext db,

IEventPublisher eventPublisher)

{

\_db = db;

\_eventPublisher = eventPublisher;

}

[HttpPost]

public async Task<IActionResult> Create(Transaction transaction)

{

transaction.Type = "Income";

if (ModelState.IsValid)

{

// Simpan ke database

\_db.Transactions.Add(transaction);

await \_db.SaveChangesAsync();

// Publish event

\_eventPublisher.Publish(new TransactionCreatedEvent(transaction));

return RedirectToAction("Index");

}

return View("Index", transaction);

}

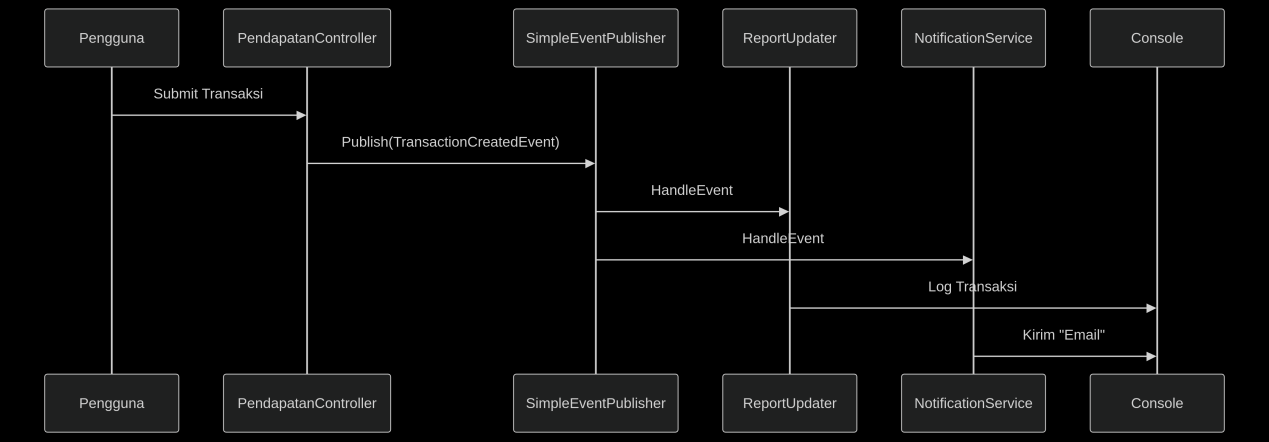
}

**Registrasi Service**

// Program.cs

builder.Services.AddScoped<NotificationService>();

**Alur Kerja Sistem**

****

**Keuntungan Implementasi**

**Decoupling**

* Controller tidak perlu tahu cara update laporan atau kirim notifikasi.
* Setiap komponen fokus pada tanggung jawabnya.

**Ekstensibilitas**

Tambahkan handler baru tanpa mengubah kode existing:

csharp

// Contoh: Audit Log

publisher.Subscribe<TransactionCreatedEvent>(e => AuditLog.Write(e.Transaction));

**Maintainability**

1. Isolasi logika bisnis ke kelas terpisah.
2. Testabilitas
3. Mudah di-mock untuk pengujian unit.
4. Peringatan & Optimasi

**Thread Safety**

Jika aplikasi multi-thread, gunakan lock untuk menghindari race condition:

csharp

// Dalam SimpleEventPublisher

private readonly object \_lock = new object();

public void Publish<TEvent>(TEvent @event)

{

lock(\_lock)

{

// ... kode publish

}

}

**Error Handling**

Tambahkan try-catch di handler untuk menghindari crash:

csharp

private void HandleTransactionCreated(TransactionCreatedEvent @event)

{

try

{

// ... logika

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

}

**Kesimpulan**

Dengan implementasi EDA custom ini:

1. Aplikasi DompetKU menjadi lebih modular.
2. Penambahan fitur baru (notifikasi, audit, dll.) tidak mengganggu kode inti.
3. Tidak ada ketergantungan pada library eksternal.

Langkah Selanjutnya:

1. Implementasi logging untuk pelacakan event.
2. Tambahkan mekanisme retry jika handler gagal.
3. Gunakan background thread untuk handler yang berat.