
Materi:

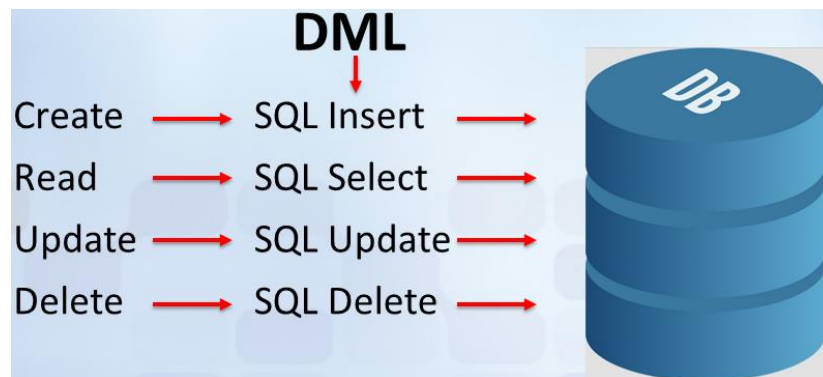
- ✓ Menggunakan Database SQLite sebagai sumber data
 - ✓ Menggunakan ADO.NET sebagai teknologi akses data
 - ✓ Implementasi Operasi CRUD Menggunakan MVC Pattern
-

Teori Singkat

Pada praktikum kali ini, kita akan mempelajari cara mengimplementasikan operasi CRUD di dalam MVC Pattern.

CRUD

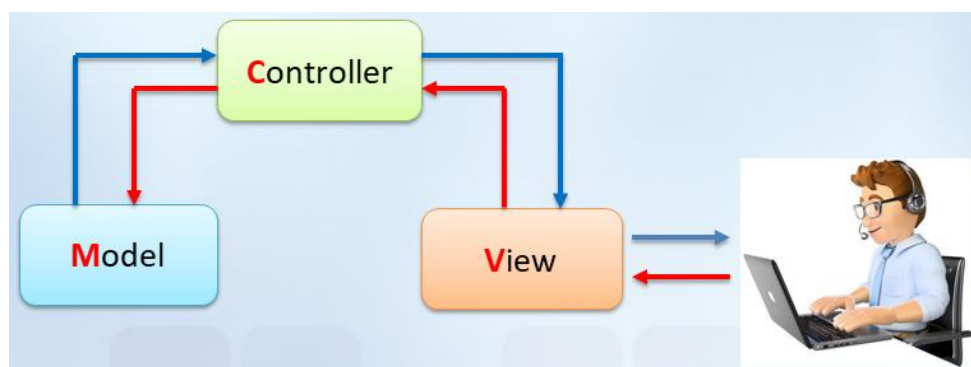
CRUD adalah istilah yang digunakan untuk operasi pengolahan data dalam sebuah aplikasi database. CRUD terdiri dari 4 operasi yaitu Create, Read, Update dan Delete.



Masing-masing operasi CRUD, menjalankan perintah SQL yang digunakan untuk memanipulasi data yang ada di dalam sebuah tabel seperti *insert*, *update*, *delete* dan *select*.

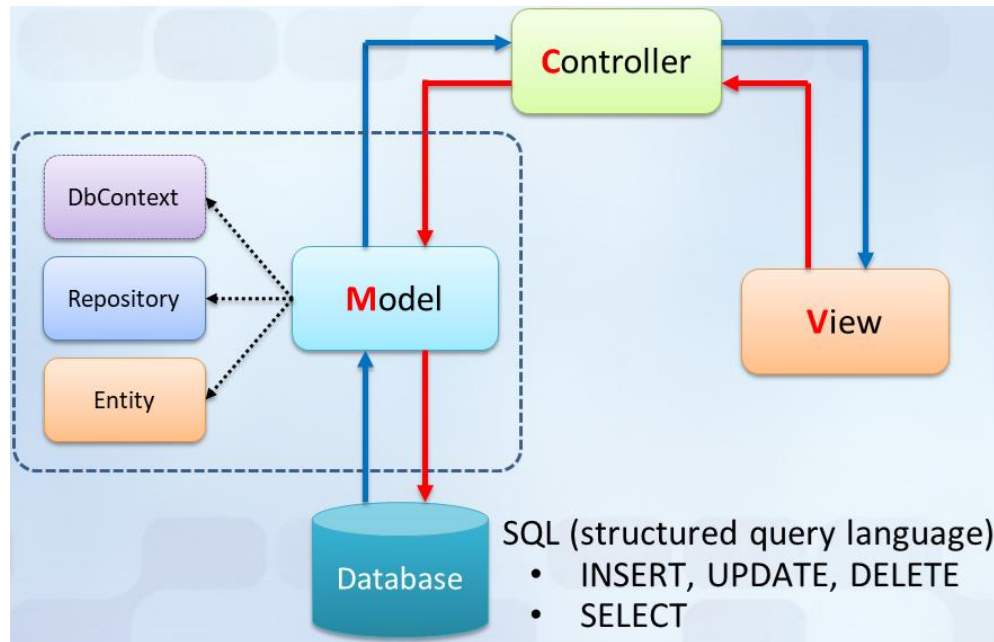
MVC Pattern

MVC Pattern adalah sebuah metode atau *design pattern* untuk membangun aplikasi dengan memisahkan data (model) dari tampilan (view) dan bagaimana cara memprosesnya (controller).



Tujuan dari dibuatnya metode ini adalah untuk memenuhi kaidah *separation of concerns* atau pemisahan kode berdasarkan fungsinya.

Komponen pertama MVC yaitu *Model* bertugas untuk melakukan komunikasi dengan berbagai sumber data seperti database dengan bantuan teknologi akses data yang ada di .NET yaitu ADO.NET.



Selain menggunakan ADO.NET, komponen Model juga didukung oleh tiga class utama untuk berkomunikasi dengan database yaitu class **DbContext**, **Repository** dan **Entity**.

Komponen berikutnya yaitu *View* bertugas untuk mengatur tampilan. Selain itu View juga bertugas untuk menerima dan mempresentasikan data kepada pengguna.

Dan terakhir yaitu komponen *Controller* merupakan bagian yang mengatur antara view dan model. Controller berfungsi untuk menerima request data dari pengguna kemudian menentukan apa yang akan diproses oleh aplikasi.

Langkah-langkah mengimplementasi operasi CRUD di dalam MVC Pattern:

✓ Membuat class **DbContext**

DbContext adalah class yang bertanggung jawab untuk berinteraksi secara langsung dengan database. Jadi class **DbContext** lah yang bertugas untuk membuat koneksi, menjalankan perintah sql seperti insert, update, delete dan select.

✓ Membuat Class Entity/Model

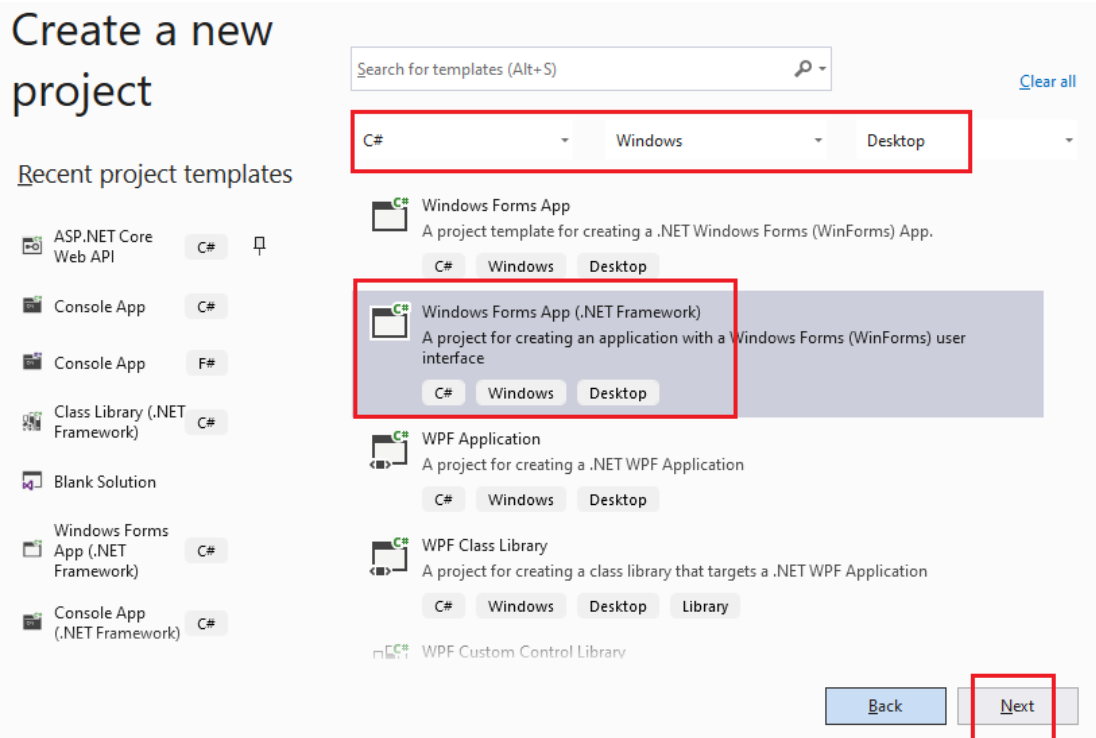
Class entity/model adalah class-class yang merupakan representasi dari tabel-tabel yang ada di dalam sebuah database.

✓ Membuat Class Repository

Class repository adalah class-class yang berisi semua kode yang berhubungan dengan operasi CRUD.

Latihan 10.1 – Membuat Project Baru

1. Jalankan aplikasi Visual Studio .NET
2. Buat project baru dengan dengan tipe Windows Forms Application



Configure your new project

Windows Forms App (.NET Framework) C# Windows Desktop

Project name

PerpustakaanAppMVC

Location

E:\Pemrograman Lanjut\Praktikum10

Solution name ⓘ

PerpustakaanAppMVC

☐ Place solution and project in the same directory

Framework

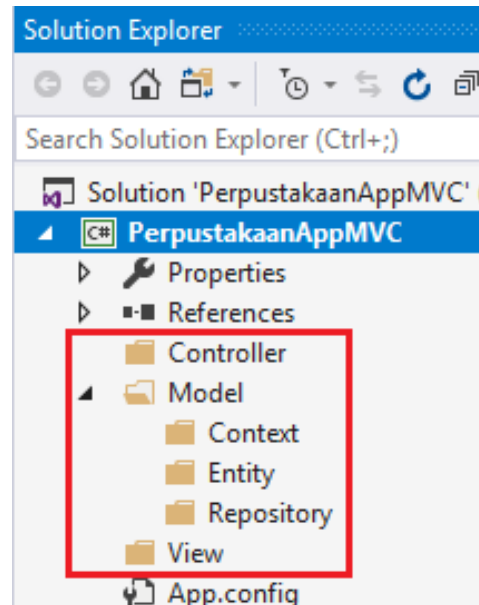
.NET Framework 4.6.1

Back Create

Isian *Project Name* diisi dengan **PerpustakaanAppMVC**. Untuk penamaan project perlu diperhatikan huruf besar kecilnya karena nama project secara default akan menjadi namespace yang akan berpengaruh terhadap penulisan kode program. Untuk isian *Location* menyesuaikan.

Latihan 10.2 – Mengatur Struktur Project untuk Mendukung MVC Pattern

Karena kita akan menggunakan MVC Pattern dalam pembuatan project ini, maka kita akan menambahkan beberapa folder yang diatur seperti gambar berikut :

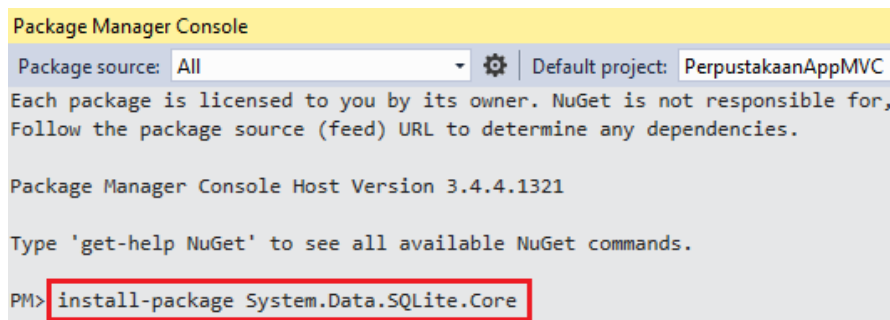


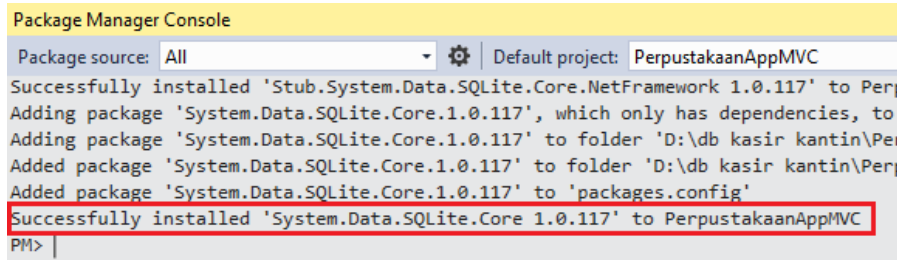
Untuk menambahkan foldernya, Anda tinggal klik kanan Project -> Add -> New Folder. Folder-folder di atas digunakan untuk mengelompokkan *source code* sesuai dengan fungsinya. Apakah kodenya berfungsi sebagai model, view atau controller.

Latihan 10.3 – Menginstall .NET Framework Data Provider for SQLite

Karena kita akan menggunakan database SQLite sebagai sumber data, maka kita harus menginstall .NET Framework Data Provider for SQLite terlebih dahulu. Berikut langkah-langkahnya:

- ✓ Klik menu tools -> NuGet Package Manager -> Package Manager Console
- ✓ Kemudian ketik perintah `install-package System.Data.SQLite.Core` pada console NuGet Package Manager



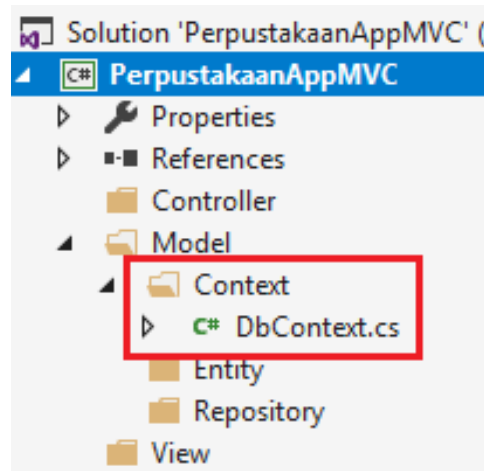


```
Package Manager Console
Package source: All | Default project: PerpustakaanAppMVC
Successfully installed 'Stub.System.Data.SQLite.Core.NetFramework 1.0.117' to PerpustakaanAppMVC
Adding package 'System.Data.SQLite.Core.1.0.117', which only has dependencies, to PerpustakaanAppMVC
Adding package 'System.Data.SQLite.Core.1.0.117' to folder 'D:\db kasir kantin\PerpustakaanAppMVC'
Added package 'System.Data.SQLite.Core.1.0.117' to folder 'D:\db kasir kantin\PerpustakaanAppMVC'
Added package 'System.Data.SQLite.Core.1.0.117' to 'packages.config'
Successfully installed 'System.Data.SQLite.Core.1.0.117' to PerpustakaanAppMVC
PM>
```

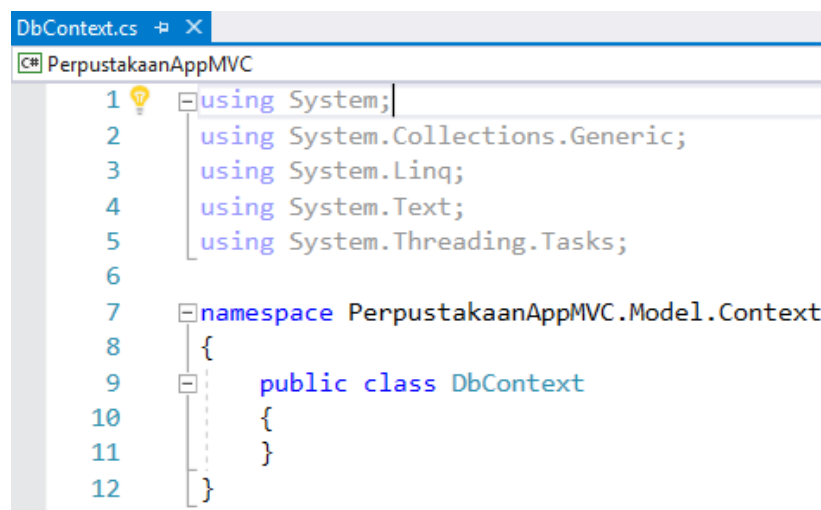
Latihan 10.4 – Menambahkan Class DbContext

Class DbContext adalah class yang bertanggung jawab untuk berinteraksi secara langsung dengan database. Jadi class DbContext lah yang bertugas untuk membuat koneksi, menjalankan perintah sql seperti insert, update, delete dan select dengan cara membungkus objek koneksi dengan menggunakan property.

1. Tambahkan class DbContext di folder Context

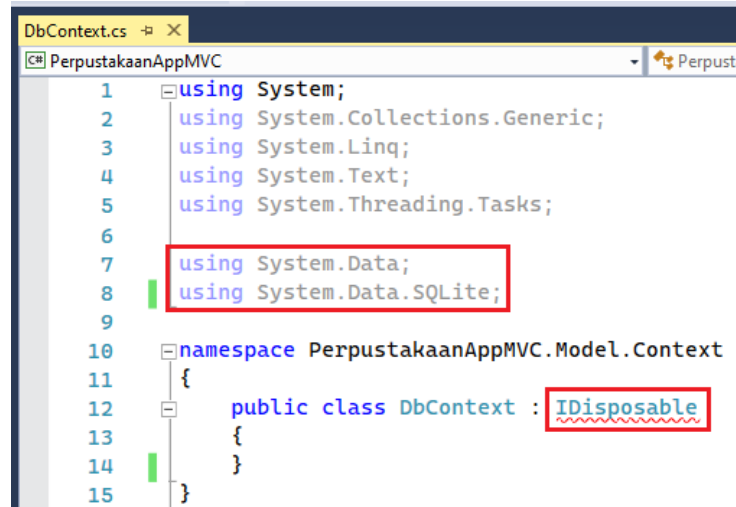


2. Setelah itu akan tampil code editor untuk class DbContext



```
DbContext.cs
C# PerpustakaanAppMVC
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace PerpustakaanAppMVC.Model.Context
8 {
9     public class DbContext
10    {
11    }
12 }
```

Kemudian tambahkan kode berikut:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 using System.Data;
8 using System.Data.SQLite;
9
10 namespace PerpustakaanAppMVC.Model.Context
11 {
12     public class DbContext : IDisposable
13     {
14     }
15 }
```

Pada kode di atas kita menambahkan dua buah namespace yaitu *System.Data* dan *System.Data.SQLite*. Namespace *System.Data.SQLite* agar kita bisa mengakses semua class dari .NET Data Provider for SQLite yang digunakan untuk koneksi ke database SQLite. Adapun fungsi dari interface *IDisposable* agar kita bisa membuat objek dari class *DbContext* menggunakan blok *using*. Dengan menggunakan cara ini, objek *DbContext* otomatis akan di hapus dari memory setelah selesai digunakan. Contoh membuat objek *DbContext* menggunakan blok *using*.

```
// membuat objek context menggunakan blok using
using (DbContext context = new DbContext())
{
    // ...
}
```

Setelah itu lengkapi kodenya seperti berikut:

```
public class DbContext : IDisposable
{
    // deklarasi private variabel / field
    private SQLiteConnection _conn;

    // deklarasi property Conn (connection), untuk menyimpan objek koneksi
    public SQLiteConnection Conn
    {
        get { return _conn ?? (_conn = GetOpenConnection()); }
    }

    // Method untuk melakukan koneksi ke database
    private SQLiteConnection GetOpenConnection()
    {
        SQLiteConnection conn = null; // deklarasi objek connection

        try // penggunaan blok try-catch untuk penanganan error
        {
            // atur ulang lokasi database yang disesuaikan dengan
            // lokasi database perpustakaan Anda
            string dbName = @"D:\Database\DbPerpustakaan.db";

            // deklarasi variabel connectionString, ref:
            https://www.connectionstrings.com/
            string connectionString = string.Format("Data
Source={0};FailIfMissing=True", dbName);

            conn = new SQLiteConnection(connectionString); // buat objek
connection
            conn.Open(); // buka koneksi ke database
        }
        // jika terjadi error di blok try, akan ditangani langsung oleh blok
catch
        catch (Exception ex)
        {
            System.Diagnostics.Debug.Print("Open Connection Error: {0}",
ex.Message);
        }

        return conn;
    }

    // Method ini digunakan untuk menghapus objek koneksi dari memory ketika
sudah tidak digunakan
    public void Dispose()
    {
        if (_conn != null)
        {
            try
            {
                if (_conn.State != ConnectionState.Closed) _conn.Close();
            }
            finally
            {
                _conn.Dispose();
            }
        }

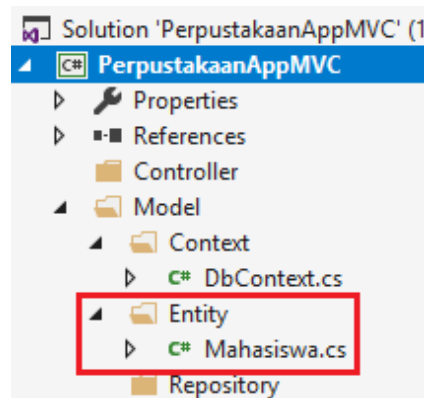
        GC.SuppressFinalize(this);
    }
}
```


Kode di atas digunakan untuk membuat koneksi ke database, dan jika berhasil objek koneksi akan tersimpan di dalam property Conn (connection).

Latihan 10.5 – Menambahkan Class Entity/Model

Class entity merupakan representasi dari tabel-tabel yang ada di database, sehingga jika kita mempunyai tabel Mahasiswa, Buku dan Sirkulasi berarti kita juga harus membuat tiga class entity yang namanya identik dengan nama tabelnya yaitu class Mahasiswa, Buku dan Sirkulasi.

1. Untuk yang pertama kita akan menambahkan class entity Mahasiswa di folder Entity.



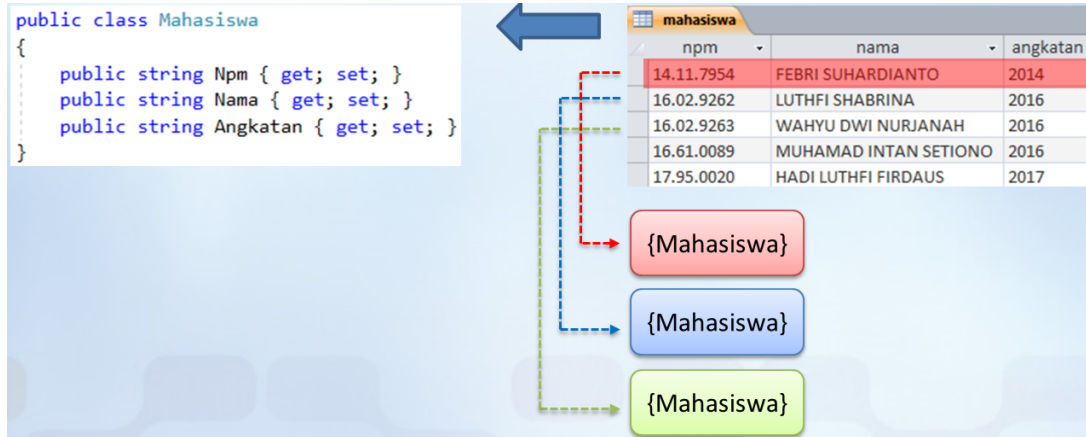
2. Kemudian lengkapi kode class entity Mahasiswa seperti berikut:

```
public class Mahasiswa
{
    public string Npm { get; set; }
    public string Nama { get; set; }
    public string Angkatan { get; set; }
}
```

Pada kode di atas, semua property yg ada di dalam class entity Mahasiswa, menyesuaikan dengan field/kolom di tabel mahasiswa yang ada di dalam database.

mahasiswa		
npm	nama	angkatan
14.11.7954	FEBRI SUHARDIANTO	2014
16.02.9262	LUTHFI SHABRINA	2016
16.02.9263	WAHYU DWI NURJANAH	2016
16.61.0089	MUHAMAD INTAN SETIONO	2016
17.95.0020	HADI LUTHFI FIRDAUS	2017

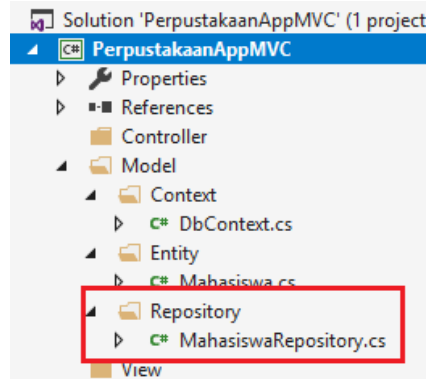
Kenapa kita harus membuat class entity Mahasiswa? Karena di dalam dunia *OOP* segala sesuatunya adalah objek. Bahkan data yang ada di dalam sebuah tabel juga di anggap objek. Sehingga kita membutuhkan objek dari class entity Mahasiswa, yang nantinya digunakan untuk menampung semua baris/record yang ada di dalam tabel mahasiswa.



Latihan 10.6 – Menambahkan Class Repository

Class repository adalah class-class yang berisi semua kode yang berhubungan dengan operasi CRUD. Aturan penamaan class repository biasanya menggunakan format: NamaClassEntityRepository. Contoh MahasiswaRepository, BukuRepository, AnggotaRepository, dan seterusnya.

1. Tambahkan class MahasiswaRepository di folder Repository.

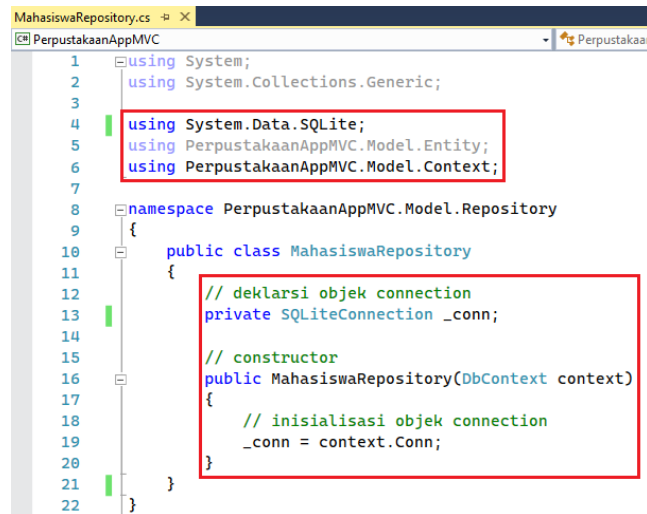


2. Setelah itu akan tampil code editor untuk class MahasiswaRepository

```
MahasiswaRepository.cs
using System.Text;
using System.Threading.Tasks;

namespace PerpustakaanAppMVC.Model.Repository
{
    public class MahasiswaRepository
    {
    }
}
```

Kemudian tambahkan kode berikut:



```
1 using System;
2 using System.Collections.Generic;
3
4 using System.Data.SQLite;
5 using PerpustakaanAppMVC.Model.Entity;
6 using PerpustakaanAppMVC.Model.Context;
7
8 namespace PerpustakaanAppMVC.Model.Repository
9 {
10     public class MahasiswaRepository
11     {
12         // deklarasi objek connection
13         private SQLiteConnection _conn;
14
15         // constructor
16         public MahasiswaRepository(DbContext context)
17         {
18             // inisialisasi objek connection
19             _conn = context.Conn;
20         }
21     }
22 }
```

Latihan 10.7 – Menambahkan Operasi/Method CRUD

1. Masih di class MahasiswaRepository, langkah berikutnya kita akan menambahkan method CRUD yang pertama yaitu *Create*. Method ini Anda tambahkan setelah *constructor* (lihat gambar di atas).

```
public int Create(Mahasiswa mhs)
{
    int result = 0;

    // deklarasi perintah SQL
    string sql = @"insert into mahasiswa (npm, nama, angkatan)
                  values (@npm, @nama, @angkatan)";

    // membuat objek command menggunakan blok using
    using (SQLiteCommand cmd = new SQLiteCommand(sql, _conn))
    {
        // mendaftarkan parameter dan mengeset nilainya
        cmd.Parameters.AddWithValue("@npm", mhs.Npm);
        cmd.Parameters.AddWithValue("@nama", mhs>Nama);
        cmd.Parameters.AddWithValue("@angkatan", mhs.Angkatan);

        try
        {
            // jalankan perintah INSERT dan tampung hasilnya ke dalam
            variabel result
            result = cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.Print("Create error: {0}", ex.Message);
        }
    }

    return result;
}
```

Pada kode di atas, method *Create* bertugas untuk mengeksekusi perintah *insert* dengan bantuan objek command. Untuk membuat objek command dibutuhkan variabel *sql* dan objek *_conn* (connection) yang di dapat dari objek *DbContext*.

Tugas 10.1:

- ✓ Lengkapi kode untuk method ***Update*** dan ***Delete***

```
public int Create(Mahasiswa mhs)
{
    int result = 0;

    // deklarasi perintah SQL
    string sql = @"insert into mahasiswa (npm, nama, angkatan)
                  values (@npm, @nama, @angkatan)";

    // membuat objek command menggunakan blok using
    using (SQLiteCommand cmd = new SQLiteCommand(sql, _conn))
    {
        // mendaftarkan parameter dan mengeset nilainya
        cmd.Parameters.AddWithValue("@npm", mhs.Npm);
        cmd.Parameters.AddWithValue("@nama", mhs>Nama);
        cmd.Parameters.AddWithValue("@angkatan", mhs.Angkatan);

        try
        {
            // jalankan perintah INSERT dan tampung hasilnya ke dalam variabel
            result = cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.Print("Create error: {0}", ex.Message);
        }
    }

    return result;
}

public int Update(Mahasiswa mhs)
{
    // TUGAS 10.1: Lengkapi kode untuk method update
}

public int Delete(Mahasiswa mhs)
{
    // TUGAS 10.1: Lengkapi kode untuk method delete
}
```

- Setelah menyelesaikan tugas 10.1 tambahkan juga method CRUD lainnya yaitu *ReadAll* dan *ReadByNama*. Method *ReadAll* digunakan untuk membaca semua data mahasiswa sedangkan *ReadByNama* digunakan untuk pencarian data mahasiswa berdasarkan nama.

```
public List<Mahasiswa> ReadAll()
{
    // membuat objek collection untuk menampung objek mahasiswa
    List<Mahasiswa> list = new List<Mahasiswa>();

    try
    {
        // deklarasi perintah SQL
        string sql = @"select npm, nama, angkatan
                        from mahasiswa
                        order by nama";

        // membuat objek command menggunakan blok using
        using (SQLiteCommand cmd = new SQLiteCommand(sql, _conn))
        {
            // membuat objek dtr (data reader) untuk menampung result set
            (hasil perintah SELECT)
            using (SQLiteDataReader dtr = cmd.ExecuteReader())
            {
                // panggil method Read untuk mendapatkan baris dari result
                set
                while (dtr.Read())
                {
                    // proses konversi dari row result set ke object
                    Mahasiswa mhs = new Mahasiswa();
                    mhs.Npm = dtr["npm"].ToString();
                    mhs>Nama = dtr["nama"].ToString();
                    mhs.Angkatan = dtr["angkatan"].ToString();

                    // tambahkan objek mahasiswa ke dalam collection
                    list.Add(mhs);
                }
            }
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.Print("ReadAll error: {0}", ex.Message);
    }

    return list;
}
```

```
public List<Mahasiswa> ReadByNama(string nama)
{
    // membuat objek collection untuk menampung objek mahasiswa
    List<Mahasiswa> list = new List<Mahasiswa>();

    try
    {
        // deklarasi perintah SQL
        string sql = @"select npm, nama, angkatan
                        from mahasiswa
                        where nama like @nama
                        order by nama";

        // membuat objek command menggunakan blok using
        using (SQLiteCommand cmd = new SQLiteCommand(sql, _conn))
        {
            // mendaftarkan parameter dan mengeset nilainya
            cmd.Parameters.AddWithValue("@nama", string.Format("%{0}%",
nama));

            // membuat objek dtr (data reader) untuk menampung result set
            (hasil perintah SELECT)
            using (SQLiteDataReader dtr = cmd.ExecuteReader())
            {
                // panggil method Read untuk mendapatkan baris dari result
set
                while (dtr.Read())
                {
                    // proses konversi dari row result set ke object
                    Mahasiswa mhs = new Mahasiswa();
                    mhs.Npm = dtr["npm"].ToString();
                    mhs>Nama = dtr["nama"].ToString();
                    mhs.Angkatan = dtr["angkatan"].ToString();

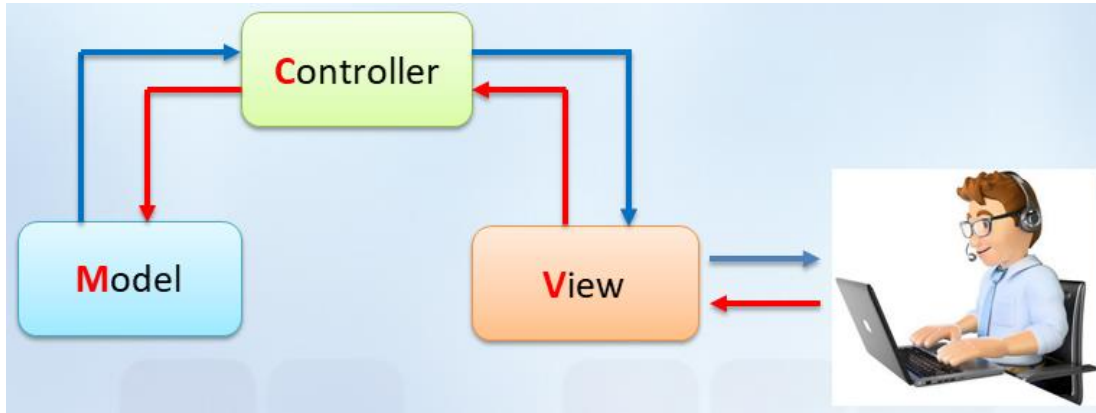
                    // tambahkan objek mahasiswa ke dalam collection
                    list.Add(mhs);
                }
            }
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.Print("ReadByNama error: {0}",
ex.Message);
    }

    return list;
}
```

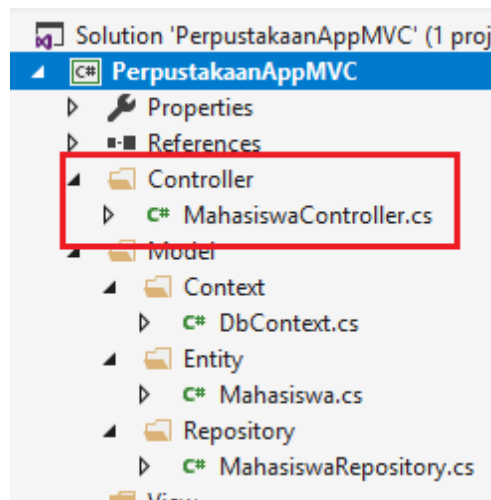
Sampai di sini penulisan kode untuk class MahasiswaRepository sudah selesai.

Latihan 10.8 – Menambahkan Class Controller

Class controller adalah class yang berfungsi untuk menerima request data dari view dan menentukan class repository (model) mana yang bertugas untuk handle request tersebut.



1. Tambahkan class MahasiswaController di folder Controller.



Setelah itu akan tampil code editor untuk class MahasiswaController.

```
MahasiswaController.cs
PerpustakaanAppMVC

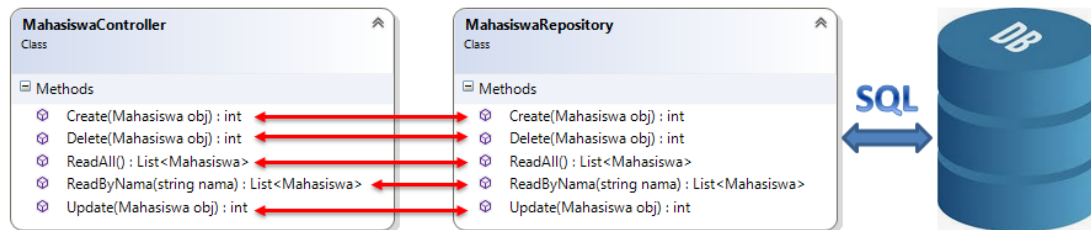
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace PerpustakaanAppMVC.Controller
8  {
9      public class MahasiswaController
10     {
11     }
12 }
```

Kemudian tambahkan kode berikut:

```
MahasiswaController.cs
PerpustakaanAppMVC

5  using System.Threading.Tasks;
6
7  using System.Windows.Forms;
8  using PerpustakaanAppMVC.Model.Entity;
9  using PerpustakaanAppMVC.Model.Repository;
10 using PerpustakaanAppMVC.Model.Context;
11
12 namespace PerpustakaanAppMVC.Controller
13 {
14     public class MahasiswaController
15     {
16         // deklarasi objek Repository untuk menjalankan operasi CRUD
17         private MahasiswaRepository _repository;
18     }
19 }
```

Class MahasiswaController juga mempunyai method yang sama seperti class MahasiswaRepository, perbedaannya hanya diimplementasi saja. Untuk method CRUD yang ada di class MahasiswaRepository semua prosesnya berhubungan langsung dengan database, sedangkan method CRUD yang ada di class MahasiswaController hanya melakukan validasi data dan memanggil method CRUD miliknya class MahasiswaRepository.



2. Masih di class MahasiswaController, kita akan menambahkan method CRUD yang pertama yaitu *Create*. Method ini Anda tambahkan setelah deklarasi field *_repository*.


```
public int Create(Mahasiswa mhs)
{
    int result = 0;

    // cek npm yang diinputkan tidak boleh kosong
    if (string.IsNullOrEmpty(mhs.Npm))
    {
        MessageBox.Show("NPM harus diisi !!!", "Peringatan",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return 0;
    }

    // cek nama yang diinputkan tidak boleh kosong
    if (string.IsNullOrEmpty(mhs>Nama))
    {
        MessageBox.Show("Nama harus diisi !!!", "Peringatan",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return 0;
    }

    // cek angkatan yang diinputkan tidak boleh kosong
    if (string.IsNullOrEmpty(mhs.Angkatan))
    {
        MessageBox.Show("Angkatan harus diisi !!!", "Peringatan",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return 0;
    }

    // membuat objek context menggunakan blok using
    using (DbContext context = new DbContext())
    {
        // membuat objek class repository
        _repository = new MahasiswaRepository(context);

        // panggil method Create class repository untuk menambahkan data
        result = _repository.Create(mhs);
    }

    if (result > 0)
    {
        MessageBox.Show("Data mahasiswa berhasil disimpan !", "Informasi",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Data mahasiswa gagal disimpan !!!", "Peringatan",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }

    return result;
}
```

Pada kode di atas, sebelum data disimpan terlebih dulu dilakukan validasi data, jika ok baru data disimpan ke database dengan memanggil method *Create* dari objek class *MahasiswaRepository*.

Tugas 10.2:

- ✓ Lengkapi kode untuk method **Update** dan **Delete**

```
public int Update(Mahasiswa mhs)
{
    // TUGAS 10.2: Lengkapi kode untuk method update
}

public int Delete(Mahasiswa mhs)
{
    // TUGAS 10.2: Lengkapi kode untuk method delete
}
```

3. Setelah menyelesaikan tugas 10.2 tambahkan juga method CRUD lainnya yaitu *ReadByNama* dan *ReadAll*.

```
public List<Mahasiswa> ReadByNama(string nama)
{
    // membuat objek collection
    List<Mahasiswa> list = new List<Mahasiswa>();

    // membuat objek context menggunakan blok using
    using (DbContext context = new DbContext())
    {
        // membuat objek dari class repository
        _repository = new MahasiswaRepository(context);

        // panggil method GetByNama yang ada di dalam class repository
        list = _repository.ReadByNama(nama);
    }

    return list;
}

public List<Mahasiswa> ReadAll()
{
    // membuat objek collection
    List<Mahasiswa> list = new List<Mahasiswa>();

    // membuat objek context menggunakan blok using
    using (DbContext context = new DbContext())
    {
        // membuat objek dari class repository
        _repository = new MahasiswaRepository(context);

        // panggil method GetAll yang ada di dalam class repository
        list = _repository.ReadAll();
    }

    return list;
}
```

Selesai 😊

Kamarudin, M.Kom
<http://coding4ever.net/>
<https://github.com/rudi-krsoftware/open-retail>