Entity Framework 2:2

Info

Reality Check

- Kraftvaerk mandag, den 14. november kl. 8:30 i Auditorium A

Ennova undersøgelsen

- http://responserate.ennova.dk/?STU2016-ErhvervsakademiAarhus/nflmnjxbkpod&ID=2005

Today's Agenda

- More About Entity Framework
  - Code First Migrations
  - Fluent API
  - SQL inspection
  - Stored procedures
  - Repository for Data Access
- Exercises

Database behaviour

- The **main class** that coordinates Entity Framework functionality for a given data model is the **`DbContext`** class

```
public class EmployeeContext : DbContext {
    public EmployeeContext() : base("EmployeeContext") {}
    public DbSet<Employee> Employees { get; set; }
}
```

- The Entity Framework can automatically create (or drop and re-create) a database for you when the application runs.
- The default behavior is to **create a database** only **if it doesn't exist** – and **throw an exception** if **the model has changed** and the database already exists.

The **Seed** method

- You can write a `Seed` method that the Entity Framework automatically calls after creating the database in order to populate it with test data

```
public class MbmStoreInitializer : System.Data.Entity.DropCreateDatabaseIfModelChanges<MbmStoreContext> {
    protected override void Seed(MbmStoreContext context) {

        var customers = new List<Customer> {
            new Customer{FirstMidName="Carson",LastName="Alexander", BirthDatDate=DateTime.Parse(”1985-09-01")},
            new Customer {FirstMidName="Meredith",LastName="Alonso",BirthDate=DateTime.Parse(”1992-09-01")}
        }
        Customers.ForEach(s => context.Customers.Add(s));
        context.SaveChanges();
    }
}
```

Options
- `CreateDatabaseIfNotExists` (default)
- `DropCreateDatabaseAlways`
- `DropCreateDatabaseIfModelChanges`

Register the seed method in `web.config`

```
<appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
    <add key="DatabaseInitializerForType
          MbmStore.DAL.MbmStoreContext, MbmStore"
          value="MbmStore.DAL.MbmStoreInitializer, MbmStore" />
  </appSettings>
```

Code First Migrations

- Drop/re-create gives you **a rough initial model with data** to work with

- Migrations (since EF 4.31, 2012) give you more **fine grained** control
  - You can make incremental changes to the DB structure without drop/recreate
  - You can reset test data to an initial state and at the same time respect updates made to certain data

Steps to take to enable migrations
(instead of drop/re-create)

1. Uncomment or delete the **`Initializer`** element in *Web.config* file
2. Change the name of the database (*Web.config*)
3. NuGet Package Manager

```
PM> enable-migrations
PM> add-migration InitialCreate
PM> update-database
...
PM> add-migration EmployeeTableCreate
PM> update-database
```

A Migrations folder is created when you enable migrations

Migrations > Configuration.cs

```
internal sealed class Configuration :
DbMigrationsConfiguration<Lesson10_solution.DAL.MbmStoreContext> {

    public Configuration() {
        AutomaticMigrationsEnabled = false;
    }

     protected override void Seed(Lesson10_solution.DAL.MbmStoreContext context){
        //  This method will be called after migrating to the latest version.

        var Prices = new List<Price> {
            new Price {Specie="Bird spider", PricePerDay=90.00M},
            new Price {Specie="Canary", PricePerDay=60.00M},
            new Price {Specie="Cat", PricePerDay=140.00M},
        };

        Prices.ForEach(p => context.Prices.AddOrUpdate(p)); // avoid duplication
        context.SaveChanges();
    }
}
```

The **Seed** method is called when the database is **created** and every time the database schema is **updated** after a data model change.

Example: Add `CreatedDate` attribute

```
public class Customer {
   public int ID { get;  set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
   public string Address { get; set; }
   public string City { get; set; }
   public string Zip { get; set; }
   public string Phone { get; set; }
   public string Email { get; set; }
   public DateTime CreatedDate { get; set; }
}
```

Add migration

Package Manager Console

Package source: nuget.org | Default project: Lesson10_solution

```
Package Manager Console Host Version 2.8.50313.46

Type 'get-help NuGet' to see all available NuGet commands.

PM> add-migration CustomerCreatedDate
```

200 %

Data Tools Operations | Error List | Output | Package Manager Console

The migration file:

(201410230844420_CustomerCreatedDate.cs)

```
6   public partial class CustomerCreatedDate : DbMigration
7   {
8       public override void Up()
9       {
10          AddColumn("dbo.Customer", "CreatedDate",
11              c => c.DateTime(nullable: false, defaultValueSql: "GETUTCDATE()"));
12      }
13
14      public override void Down()
15      {
16          DropColumn("dbo.Customer", "CreatedDate");
17      }
18  }
```

Executing the migration code

Package Manager Console

Package source: nuget.org Default project: lesson09_examples

```
want to include in this migration, then you can re-scaffold it by running 'Add-Migration CustomerCr
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [201511031057487_CustomerCreatedDate].
Applying explicit migration: 201511031057487_CustomerCreatedDate.
Running Seed method.
PM>
```

Execute the migration

```
PM> update-database
```

Update | Script File: dbo.Customer.sql

	Name	Data Type	Allow Nulls	Default
	ID	int	☐	
	FirstName	nvarchar(MAX)	☑	
	LastName	nvarchar(MAX)	☑	
	Address	nvarchar(MAX)	☑	
	City	nvarchar(MAX)	☑	
	Zip	nvarchar(MAX)	☑	
	Phone	nvarchar(MAX)	☑	
	Email	nvarchar(MAX)	☑	
	CreatedDate	datetime	☐	(getutcdate())
			☐	

Rollback migrations

```
PM> Get-Migrations

Retrieving migrations that have been applied to the target database.
201208012131302_Add-SystemCategory
201207311827468_CategoryIdIsLong
201207232247409_AutomaticMigration
201207211340509_AutomaticMigration
201207200025294_InitialCreate

PM> Update-Database -TargetMigration:"CategoryIdIsLong"
```

Rollback all

```
update-database -target:0
```

http://stackoverflow.com/questions/11904571/ef-migrations-rollback-last-applied-migration

Exercises

Fluent API

Fluent API

- Property Mapping
- Type Mapping
- Configuring Relationships

The file to configure: **DBContext** Class

```
namespace Lesson10_solution.DAL {
    public class MbmStoreContext : DbContext {
        public MbmStoreContext() : base("MbmStoreContext") {
        }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<Invoice> Invoices { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder) {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            // Fluent API code
        }
}
```

Property Mapping (examples)

Primary key

```
modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorId);
```

Composite Primary Key

```
modelBuilder.Entity<Department>()
.HasKey(t => new { t.DepartmentId, t.Name });
```

Maximum length of a property

```
modelBuilder.Entity<Department>()
        .Property(t => t.Name).HasMaxLength(50);
```

Required property

```
modelBuilder.Entity<Department>()
        .Property(t => t.Name).IsRequired();
```

Type Mapping (examples)

Mapping an Entity Type to a Specific Table in the Database

```
modelBuilder.Entity<Department>()
.ToTable("t_Department");
```

Not to Map a CLR Entity Type to a Table in the Database

```
modelBuilder.Ignore<OnlineCourse>();
```

Mapping Properties of an Entity Type to Multiple Tables in the Database (Entity Splitting)

```
modelBuilder.Entity<Department>()
.Map(m => {
    m.Properties(t => new { t.DepartmentID, t.Name });
    m.ToTable("Department");
})
.Map(m =>
{
    m.Properties(t => new { t.DepartmentID, t.Administrator, t.Budget
});
    m.ToTable("DepartmentDetails");
});
```

Configuring Relationships: One-to-Many

```
modelBuilder.Entity<Course>()
        .HasRequired(c => c.Department)
        .WithMany(d => d.Courses)
        .HasForeignKey(c => c.SomeDepartmentID);
```

Configuring Relationships: Many-to-Many

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses)
```

Configuring Relationships: Many-to-Many

- specify the join table name and the names of the columns

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses)
    .Map(m =>
    {
        m.ToTable("CourseInstructor");
        m.MapLeftKey("CourseID");
        m.MapRightKey("InstructorID");
    });
```

Configuring Relationships: Enabling Cascade Delete

```
modelBuilder.Entity<Course>()
    .HasRequired(t => t.Department)
    .WithMany(t => t.Courses)
    .HasForeignKey(d => d.DepartmentID)
    .WillCascadeOnDelete(true);
```

Read more

- Configuring/Mapping Properties and Types with the Fluent API
  http://msdn.microsoft.com/en-us/data/jj591617.aspx
- Configuring Relationships with the Fluent API
  http://msdn.microsoft.com/en-us/data/jj591620.aspx
- Fluent API in Code-First
  http://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx

Inspect SQL

Insert a breakpoint and run in debug mode

Inspect SQL

```
14  public class CustomerController : Controller
15  {
16      private lesson09_examplesContext db = new lesson09_examplesContext();
17
18      // GET: Customer
19      public ActionResult Index()
20      {
21          return View(db.Customers.ToList());
22  db.Customers {SELECT [Extent1].[CustomerId] AS [CustomerId], [Extent1].[Name] AS [Name], [Extent1].[City] AS [City], [Extent1].[Gender] AS [Gender] FROM [dbo].[Customer] AS [Extent1]}
23
```

The SQL

```
db.Customers = {SELECT
    [Extent1].[CustomerId] AS [CustomerId],
    [Extent1].[Name] AS [Name],
    [Extent1].[City] AS [City],
    [Extent1].[Gender] AS [Gender]
    FROM [dbo].[Customer] AS [Extent1]}
```

Log and Inspect SQL (since EF 6.1)

Add this to the `entityFramework` node of the `web.config` file:

```
<interceptors>
   <interceptor type="System.Data.Entity.Infrastructure
           .Interception.DatabaseLogger, EntityFramework">
      <parameters>
         <parameter value="C:\Temp\LogSQLOutput.txt"/>
      </parameters>
   </interceptor>
</interceptors>
```

Inspect SQL: `C:\Temp\LogSQLOutput.txt`

```
Opened connection at 04-11-2014 17:05:06 +01:00
SELECT
    [Extent1].[ID] AS [ID],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[Address] AS [Address],
    [Extent1].[City] AS [City],
    [Extent1].[Zip] AS [Zip],
    [Extent1].[Phone] AS [Phone],
    [Extent1].[Email] AS [Email],
    [Extent1].[CreatedDate] AS [CreatedDate]
    FROM [dbo].[Customer] AS [Extent1]
-- Executing at 04-11-2014 17:05:06 +01:00
-- Completed in 4 ms with result: SqlDataReader

Closed connection at 04-11-2014 17:05:06 +01:00
```

Use stored procedures

For inserting, updating, and deleting

Step 1: Add Stored Procedure to an Entity

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {

    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    modelBuilder.Entity<Customer>().MapToStoredProcedures();
}
```

Step 2: Create migration

```
PM>add-migration CustomerSP
```

An migration with scripts for stored procedures for **INSERT**, **UPDATE** and **DELETE** are created

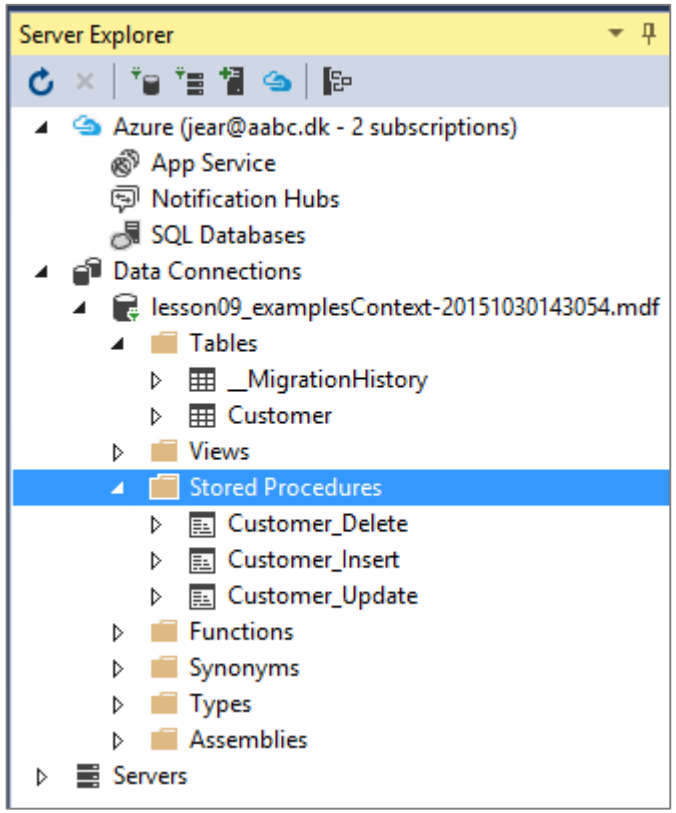
```
public override void Up()
{
    CreateStoredProcedure(
        "dbo.Customer_Insert",
        p => new
            {
                Name = p.String(),
                City = p.String(),
                Gender = p.Int(),
            },
        body:
            @"INSERT [dbo].[Customer]([Name], [City], [Gender])
              VALUES (@Name, @City, @Gender)

              DECLARE @CustomerId int
              SELECT @CustomerId = [CustomerId]
              FROM [dbo].[Customer]
              WHERE @@ROWCOUNT > 0 AND [CustomerId] = scope_identity()

              SELECT t0.[CustomerId]
              FROM [dbo].[Customer] AS t0
              WHERE @@ROWCOUNT > 0 AND t0.[CustomerId] = @CustomerId"
    );
```

Step 3: Create the stored procedures

```
PM>update-database
```



Work with SQL
and call stored procedures

Retrieve data from a View

```
public ActionResult SelectFromView()
    using (db) // db is the context class
    {
        // The DB View is named CustomersView
        // Each row returned is a Customer
        List<Customer> customers =
            db.Database.SqlQuery<Customer>(
            "select * from CustomersView").ToList();
    }
    return View();
}
```

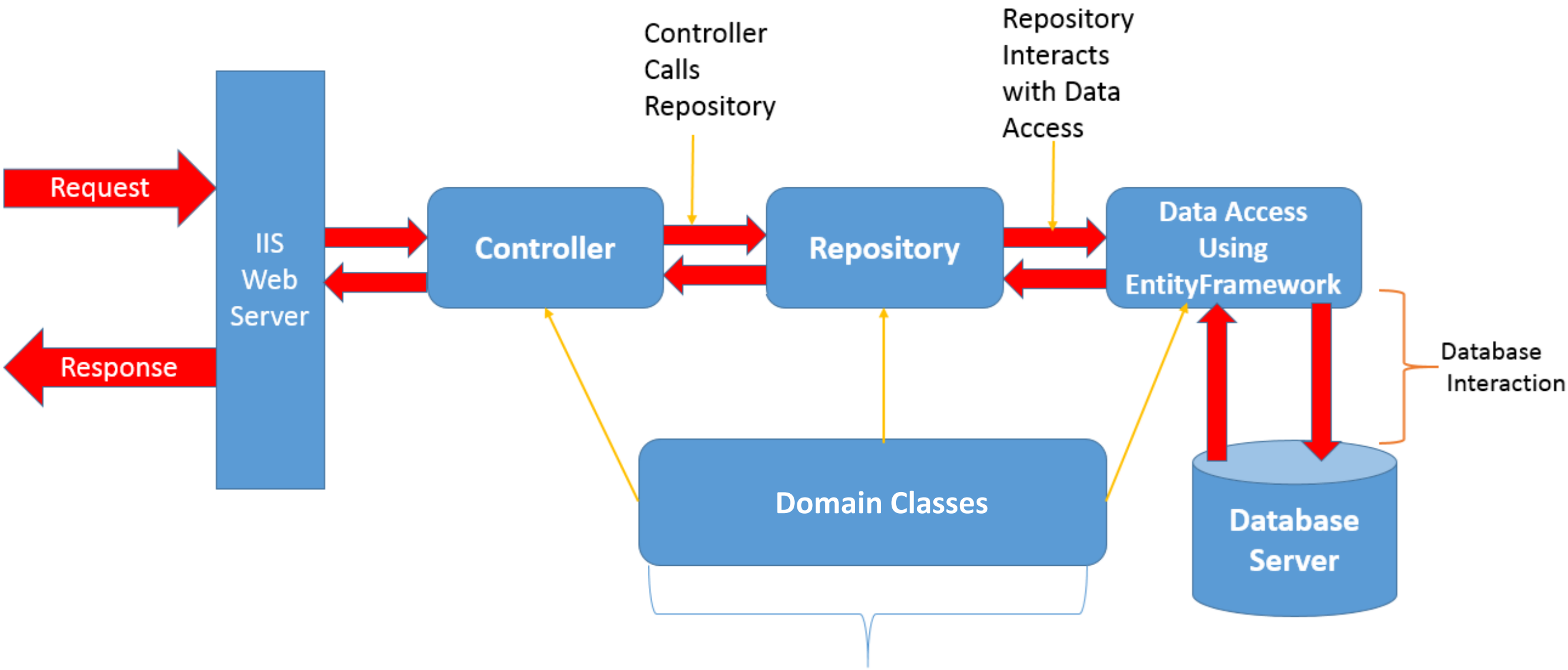
Retrieve data from a Stored Procedure

```
public ActionResult RetriveCustomersByZip() {
    List<Custumer> customers;
    using (db) { // db is the DbContext object
        // Zip is the in-parameter for the stored procedure
        SqlParameter param1 = new SqlParameter("@Zip", "8000");
        // Call the stored procedure with the parameter
        // <Customer> represents the type returned from each row
        customers = db.Database.SqlQuery<Customer>("exec
CustomersByZip @Zip", param1).ToList<Customer>();
    }
    return View(orders);
    }
```

Use ExecuteSqlCommand for Insert, Update, and Delete Stored Procedures

```
public ActionResult CreateCustomer(Customer customer) {
   using (db) {
      SqlParameter param1 = new SqlParameter("@Firstname",
customer.Firstname);
      SqlParameter param2 = new SqlParameter("@Lastname",
customer.Firstname);
      // Call the stored procedure with parameters
      db.Database.ExecuteSqlCommand("exec CreateCustomer
      @Firstname, @Lastname", param1,param2);
   }
   return View(customer);
}
```

Repository



Domain Classes used by all Layers

BookRepository

Example

The Interface

```
namespace MbmStore.DAL
{
    public interface IBookRepository
    {
        IEnumerable<Book> GetBookList();
        Book GetBookById(int id);
        void SaveBook(Book book);
        Book DeleteBook(int bookId);
    }
}
```

The Repository Class (fragment)

```
namespace MbmStore.DAL {
  public class EFBookRepository : IBookRepository {

  private MbmStoreContext db = new MbmStoreContext();

  public IEnumerable<Book> GetBookList() {
      return db.Books.ToList();
  }


  public Book GetBookById(int id) {
      return db.Books.Find(id);
  }
…
```

The Repository Class (SaveBook)

```
public void SaveBook(Book book) {
      if (book.ProductId == 0) {
          book.CreatedDate = DateTime.Now;
          db.Books.Add(book);
          db.SaveChanges();
      }
      else  {
          db.Entry(book).State = EntityState.Modified;
          db.Entry(book).Property(c => c.CreatedDate).IsModified = false;
          db.SaveChanges();
      }
}
```

Using the repository class

```
namespace MbmStore.Areas.Admin.Controllers {
    public class BookController : Controller {
        private IBookRepository repo = new EFBookRepository();

        // GET: Admin/Book
        public ActionResult Index() {
            return View(repo.GetBookList());
        }
  . . .
```

Tip on this error:

Cannot attach the file *.mdf as database

```
 <add name="MbmStoreContext" connectionString="Data
Source=(localdb)\v11.0; Initial Catalog=MbmStore;
Integrated Security=True; MultipleActiveResultSets=True;
AttachDbFilename=|DataDirectory|MbmStore.mdf"
providerName="System.Data.SqlClient" />
```

->

```
<add name="MbmStoreContext" connectionString="Data
Source=(localdb)\v11.0; Integrated Security=True;
MultipleActiveResultSets=True;
AttachDbFilename=|DataDirectory|MbmStore.mdf"
providerName="System.Data.SqlClient" />
```

Exercises

Continuing working on **MbmStore** project