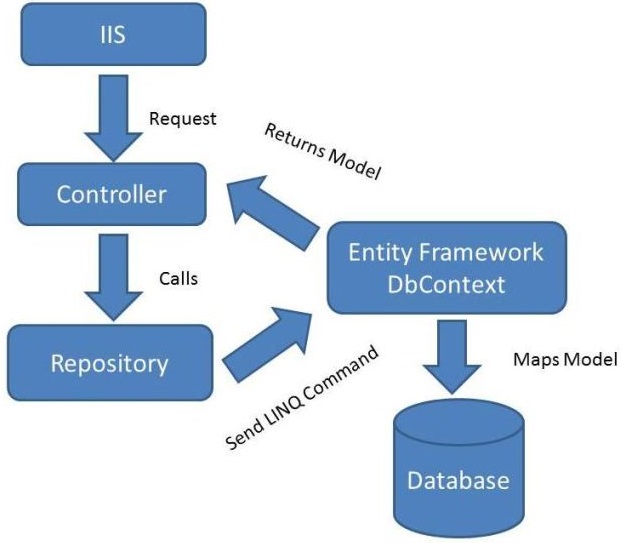
**Overview of the Repository Pattern**

http://www.codeproject.com/Articles/644605/CRUD-Operations-Using-the-Repository-Pattern-in-MV

The Repository pattern is intended to create an abstraction layer between the data access layer and the business logic layer of an application. It is a data access pattern that prompts a more loosely coupled approach to data access. We create the data access logic in a separate class, or set of classes, called a repository, with the responsibility of persisting the application's business model.



Examples with code demo

In the DAL folder create an IBookRepository interface that has the filename IBookRepository.cs. This interface code declares a typical set of CRUD methods, including two read methods; one that returns all Book entity sets, and one that finds a single Book entity by ID.

using System;

using System.Collections.Generic;

using BookStore.Models;

namespace BookStore.DAL

{

public interface IBookRepository : IDisposable

{

IEnumerable<Book> GetBooks();

Book GetBookByID(int bookId);

void InsertBook(Book book);

void DeleteBook(int bookID);

void UpdateBook(Book book);

void Save();

}

}

In the DAL folder, create a class file named BookRepository.cs. The class file implements the IBookRepository interface and the IBookRepository inherits the IDisposable interface so the IDisposable interface is indirectly implemented by the BookRespository class.

using System;

using System.Collections.Generic;

using System.Linq;

using BookStore.Models;

using System.Data;

namespace BookStore.DAL

{

public class BookRepository : IBookRepository

{

private BookContext \_context;

public BookRepository(BookContext bookContext)

{

this.\_context = bookContext;

}

public IEnumerable<book> GetBooks()

{

return \_context.Books.ToList();

}

public Book GetBookByID(int id)

{

return \_context.Books.Find(id);

}

public void InsertBook(Book book)

{

\_context.Books.Add(book);

}

public void DeleteBook(int bookID)

{

Book book = \_context.Books.Find(bookID);

\_context.Books.Remove(book);

}

public void UpdateBook(Book book)

{

\_context.Entry(book).State = EntityState.Modified;

}

public void Save()

{

\_context.SaveChanges();

}

private bool disposed = false;

protected virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

\_context.Dispose();

}

}

this.disposed = true;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

}

using System.Data;

using System.Linq;

using System.Web.Mvc;

using BookStore.DAL;

using BookStore.Models;

We create an instance of the Book repository interface in the Book Controller and initialize the book repository in the constructor of Book Controller (*BookController.cs*) as in the following:

Hide   Copy Code

private IBookRepository \_bookRepository;

public BookController()

{

this.\_bookRepository = new BookRepository(new BookContext());

}

public ActionResult Index()

{

var books = from book in \_bookRepository.GetBooks()

select book;

return View(books);

}

public ViewResult Details(int id)

{

Book student = \_bookRepository.GetBookByID(id);

return View(student);

}

public ActionResult Create()

{

return View(new Book());

}

[HttpPost]

public ActionResult Create(Book book)

{

try

{

if (ModelState.IsValid)

{

\_bookRepository.InsertBook(book);

\_bookRepository.Save();

return RedirectToAction("Index");

}

}

catch (DataException)

{

ModelState.AddModelError("", "Unable to save changes. " +

"Try again, and if the problem persists see your system administrator.");

}

return View(book);

}

public ActionResult Edit(int id)

{

Book book = \_bookRepository.GetBookByID(id);

return View(book);

}

[HttpPost]

public ActionResult Edit(Book book)

{

try

{

if (ModelState.IsValid)

{

\_bookRepository.UpdateBook(book);

\_bookRepository.Save();

return RedirectToAction("Index");

}

}

catch (DataException)

{

ModelState.AddModelError("", "Unable to save changes. Try again, " +

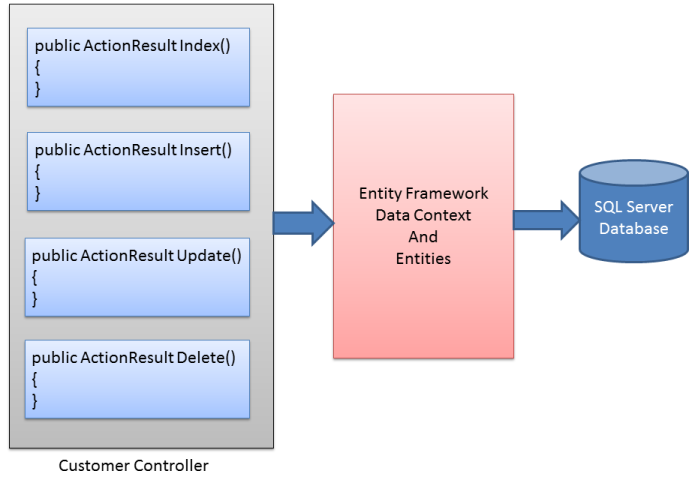
"and if the problem persists see your system administrator.");

}

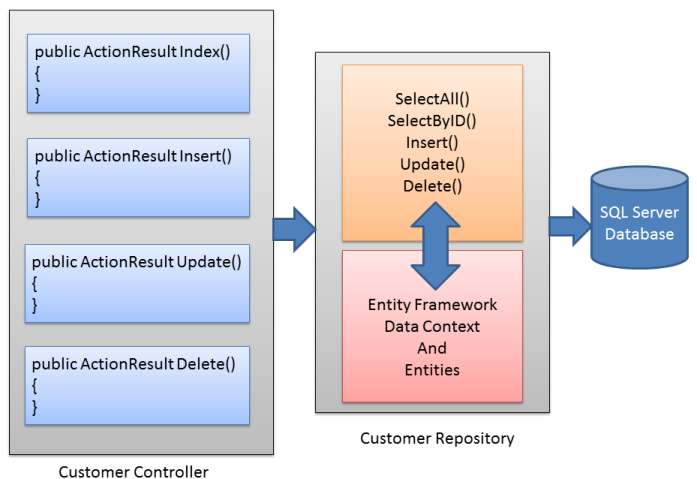
return View(book);

}

Example 2:



pseudo-database would do the trick. With a repository introduced, the above figure can be changed to:



public interface ICustomerRepository

{

IEnumerable<Customer> SelectAll();

Customer SelectByID(string id);

void Insert(Customer obj);

void Update(Customer obj);

void Delete(string id);

void Save();

}

public class CustomerRepository:ICustomerRepository

{

private NorthwindEntities db = null;

public CustomerRepository()

{

this.db = new NorthwindEntities();

}

public CustomerRepository(NorthwindEntities db)

{

this.db = db;

}

public IEnumerable<Customer> SelectAll()

{

return db.Customers.ToList();

}

public Customer SelectByID(string id)

{

return db.Customers.Find(id);

}

public void Insert(Customer obj)

{

db.Customers.Add(obj);

}

public void Update(Customer obj)

{

db.Entry(obj).State = EntityState.Modified;

}

public void Delete(string id)

{

Customer existing = db.Customers.Find(id);

db.Customers.Remove(existing);

}

public void Save()

{

db.SaveChanges();

}

public class CustomerController : Controller

{

**private ICustomerRepository repository = null;**

**public CustomerController()**

**{**

**this.repository = new CustomerRepository();**

**}**

**public CustomerController(ICustomerRepository repository)**

**{**

**this.repository = repository;**

**}**

public ActionResult Index()

{

List<Customer> model =

(List<Customer>)repository.SelectAll();

return View(model);

}

public ActionResult New()

{

return View();

}

public ActionResult Insert(Customer obj)

{

repository.Insert(obj);

repository.Save();

return View();

}

public ActionResult Edit(string id)

{

Customer existing = repository.SelectByID(id);

return View(existing);

}

public ActionResult Update(Customer obj)

{

repository.Update(obj);

repository.Save();

return View();

}

public ActionResult ConfirmDelete(string id)

{

Customer existing = repository.SelectByID(id);

return View(existing);

}

public ActionResult Delete(string id)

{

repository.Delete(id);

repository.Save();

return View();

}

}

