# step to create generic repository & Unit of Work for data access

1. Add Generic Repository interface:
   1. Add an interface that accepts generic type
      1. public interface IRepo<T> where T: class
   2. Add interface methods for CRUD
      1. void Add(T obj); // Create
      2. IEnumerable<T> GetAll(); //Read
      3. T getById(object id); //Read
      4. IQueryable<T> Find(Expression<Func<T,bool>> predicate); //Read (linq expression for dynamic search)
      5. void Update(T obj); //Update
      6. void Delete(Object id); //Delete
2. Add Concrete Implementation for Repository interface
   1. public class Repository<T> : IRepo<T> where T:class
   2. private fields for context and entity
   3. implement constructor that takes DbContext as parameter and assigns to context field
   4. implement DbSet that loads type T by calling context.Set<T>() method and assign into entity
   5. implement interface methods for CRUD on entity
3. Add Unit of Work Interface
   1. public interface IUnitOfWork {
      1. Commit();
   2. }
4. Add Concrete implementation for IUnitOfWork
   1. public class UnitOfWork : IUnitOfWork, IDisposable
   2. create a new instance of DbContext
   3. create fields for Entity Repository & add setters (gen one up with context if one not found)
   4. implement commit
   5. implement disposable (dispose of context)

# AutoMapper

Automapper allows you to map values from one type to other for matching fields. You can also add custom logic to assign values to destination type fields.

1. Install NuGet package.
2. Initialize mapper by creating maps on application start.   
    Mapper.Initialize(cfg =>

{

cfg.CreateMap<Type1, Type2>();

})

1. DestObject = Mapper.Map<soucetype, desttype>(sourceobject)

# Predicate Builder (add it project for dynamic linq expression)

var pb = PredicateBuilder.True<TYPE>(); //create a variable to hold dynamic criteria

pb = pb.And(c => c.FirstName.ToLower().StartsWith(name) || c.LastName.ToLower().StartsWith(name)); //add to predicate

Expression<Func<Contact, bool>> Expr = pb; //assign to linq expression

Pass that to EF repository to get results

# AJAX forms

NuGet Microsoft unobtrusive jquery  
Create a bundle for unobtrusive jquery and add to page  
Use Ajax.BeginForm directive point to action and controller and add ajax options for post, UpdateTargetId (a placeholder where content will be injected)  
 @using (Ajax.BeginForm("GetContacts", "Contacts", new AjaxOptions

{

HttpMethod = "POST",

UpdateTargetId = "contactsList"

}, new { id = "MyNewNameId" }))

{

Form stuff with submit button

}

<div id="contactsList"></div> //place holder

# POPUPS

Create a placeholder div for popup information and scripts to manage popup (use bootstrap modal-container)

<div id="modal-container" class="modal fade"

tabindex="-1" role="dialog">

<div class="modal-dialog">

<div class="modal-content"></div>

</div>

</div>

<script type="text/javascript">

$(function () {

// Initialize numeric spinner input boxes

//$(".numeric-spinner").spinedit();

// Initialize modal dialog

// attach modal-container bootstrap attributes to links with .modal-link class.

// when a link is clicked with these attributes, bootstrap will display the href content in a modal dialog.

$('body').on('click', '.modal-link', function (e) {

e.preventDefault();

$(this).attr('data-target', '#modal-container');

$(this).attr('data-toggle', 'modal');

});

// Attach listener to .modal-close-btn's so that when the button is pressed the modal dialog disappears

$('body').on('click', '.modal-close-btn', function () {

$('#modal-container').modal('hide');

});

//clear modal cache, so that new content can be loaded

$('#modal-container').on('hidden.bs.modal', function () {

$(this).removeData('bs.modal');

});

$('#CancelModal').on('click', function () {

return false;

});

});

$(function () {

$('#popup-submit-btn').click(function () {

$('#modal-container').modal('hide');

});

});

</script>

Add action link to open partial view in the popup container. Class assignment binds event that allows popup content to be displayed in the container.  
 @Html.ActionLink("Create New", "Create", null, new { @class = "modal-link btn btn-primary" })