New Project > ASP.NET MVC4 “CodeTalks” > Internet Application

Rename “CodeTalks” to “CodeTalks.Web”

Add 3 Class Library projects:

CodeTalks.Model

CodeTalks.Data

CodeTalks.Data.Contracts

5. ServiceLayer

5a. ITalkService

5b. TalkService

Add references:

CodeTalks.Web ->

CodeTalks.Model (none)

CodeTalks.Data.Contracts ->

* CodeTalks.Model

CodeTalks.Data ->

* CodeTalks.Data.Contracts
* CodeTalks.Model
* EntityFramework

**Solution Structure**

4 projects:

CodeTalks.Web

CodeTalks.Model

CodeTalks.Data

CodeTalks.Data.Contracts

**Model**

CodeTalks.Model

**POCO Models**

no project references

domain objects

vehicles for data

properties that define data

EF will populate them

POCO

* Plain Old Class Object
* Independent and stand alone objects
* Easy to pass around

Web API > Unit of Work > Repositories > EF > db

 1.

namespace CodeTalks.Model

{

  public class Talk

  {

    public int Id { get; set; }

    public string Title { get; set; }

    public string Body { get; set; }

    public DateTime Created { get; set; }

    public bool Flagged { get; set; }

    public ICollection<Comment> Comments { get; set; }

  }

}

namespace CodeTalks.Model

{

  public class Comment

  {

    public int Id { get; set; }

    public string Body { get; set; }

    public DateTime Created { get; set; }

    public int TalkId { get; set; }

  }

}

3.

Context – communication between code and db – conduit for crud

Entity Framework - Nuget package or Add reference

"DefaultConnection"- from security video?

namespace CodeTalks.Data

{

    public class CodeTalksContext: DbContext

    {

        public CodeTalksContext()

            : base("DefaultConnection")

        {

        }

        public DbSet<Talk> Talks { get; set; }

        public DbSet<Comment> Comments { get; set; }

    }

}

**2.**

**Interfaces**

CodeTalks.Data.Contracts

References:

* CodeTalks.Model

Just methods we need to get at data

//paging, ordering, grouping, sorting, filtering…

so IQueryable (rather than IEnumerable)

Just start with   
-GetTalks()   
-GetCommentsByTalk

Allows us to mock up a repo for testing later

namespace CodeTalks.Data.Contracts

{

  public interface ICodeTalksRepository

  {

    IQueryable<Talk> GetTalks();

    IQueryable<Talk> GetTalksIncludingComments();

IQueryable<Comment> GetCommentsByTalk(int talkId);

    bool Save();

    bool AddTalk(Talk newTalk);

    bool AddComment(Comment newComment);

  }

}

**4.**

**Repository** and **Unit of Work**

CodeTalks**.Data**

References:

* CodeTalks.Data.Contracts
* CodeTalks.Model
* EntityFramework

namespace CodeTalks.Data

{

  public class CodeTalksRepository : ICodeTalksRepository

  {

    CodeTalksContext \_ctx;

    public MessageBoardRepository(CodeTalksContext ctx)

    {

      \_ctx = ctx;

    }

    public IQueryable<Talk> GetTalks()

    {

      return \_ctx.Talks;

    }

    public IQueryable<Comment> GetCommentsByTalk(int talkId)

    {

      return \_ctx.Comments.Where(c => c.TalkId == talkId);

    }

    public bool Save()

    {

      try

      {

        return \_ctx.SaveChanges() > 0;

      }

      catch (Exception ex)

      {

        // TODO log this error

        return false;

      }

    }

    public bool AddTalk(Talk newTalk)

    {

      try

      {

        \_ctx.Talks.Add(newTalk);

        return true;

      }

      catch (Exception ex)

      {

        // TODO log this error

        return false;

      }

    }

    public IQueryable<Topic> GetTalksIncludingComments()

    {

      return \_ctx.Talks.Include("Comments");

    }

    public bool AddComment(Comment newComment)

    {

      try

      {

        \_ctx.Comments.Add(newComment);

        return true;

      }

      catch (Exception ex)

      {

        // TODO log this error

        return false;

      }

    }

  }

}

namespace CodeTalks.Data

{

  public class CodeTalksContext : DbContext

  {

    public CodeTalksContext()

      : base("DefaultConnection")

    {

      this.Configuration.LazyLoadingEnabled = false;

      this.Configuration.ProxyCreationEnabled = false;

   }

    public DbSet<Talk> Talks { get; set; }

    public DbSet<Comment> Comments { get; set; }

  }

}

Ninject

    /// <summary>

    /// Load your modules or register your services here!

    /// </summary>

    /// <param name="kernel">The kernel.</param>

    private static void RegisterServices(IKernel kernel)

    {

#if DEBUG

      kernel.Bind<IMailService>().To<MockMailService>().InRequestScope();

#else

      kernel.Bind<IMailService>().To<MailService>().InRequestScope();

#endif

      kernel.Bind<CodeTalksContext>().To<CodeTalksContext>().InRequestScope();

      kernel.Bind<ICodeTalksRepository>().To<CodeTalksRepository>().InRequestScope();

    }

5. ServiceLayer

5a. ITalkService

5b. TalkService

4.

//right click ICodeTalksRepository to implement

namespace CodeTalks.Data

{

    public class CodeTalksRepository: ICodeTalksRepository

    {

        public IQueryable<Talk> GetTalks()

        {

            throw new NotImplementedException();

        }

        public IQueryable<Comment> GetCommentsByTalk(int talkId)

        {

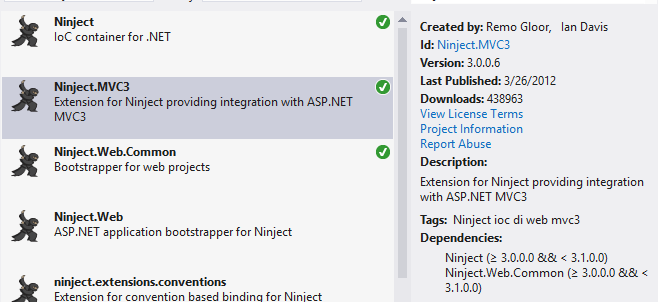
            throw new NotImplementedException();

        }

    }

}

4.1 Ninject



    /// <summary>

    /// Load your modules or register your services here!

    /// </summary>

    /// <param name="kernel">The kernel.</param>

    private static void RegisterServices(IKernel kernel)

    {

#if DEBUG

      kernel.Bind<IMailService>().To<MockMailService>().InRequestScope();

#else

      kernel.Bind<IMailService>().To<MailService>().InRequestScope();

#endif

      kernel.Bind<CodeTalksContext>().To<CodeTalksContext>().InRequestScope();

      kernel.Bind<ICodeTalksRepository>().To<CodeTalksRepository>().InRequestScope();

    }

4.2

//next… (a)

// works, but implies a new MessageBoardContext each time call GetTopics

// create and dispose

public class MessageBoardRepository : IMessageBoardRepository

{

public IQueryable<Topic> GetTopics()

{

var ctx = new MessageBoardContext();

return ctx.Topics;

// and it's disposable so…

**using (**var ctx = new MessageBoardContext()**)**

**{**

return ctx.Topics;

**}**

}

public IQueryable<Reply> GetRepliesByTopic(int topicId)

{

}

}

//next… (b)

// so create a single instance of ctx in the ctor

public class MessageBoardRepository : IMessageBoardRepository

{

**MessageBoardContext \_ctx;**

**public MessageBoardRepository()**

**{**

**\_ctx = new MessageBoardContext();**

**}**

public IQueryable<Topic> GetTopics()

{

return \_ctx.Topics;

}

public IQueryable<Reply> GetRepliesByTopic(int topicId)

{

}

}

//next… (c)

// but we'd prefer to have the ctx passed to us so we can use any one that already exists

public class MessageBoardRepository : IMessageBoardRepository

{

MessageBoardContext \_ctx;

public MessageBoardRepository(**MessageBoardContext ctx**)

{

return \_ctx = **ctx**;

}

public IQueryable<Topic> GetTopics()

{

}

public IQueryable<Reply> GetRepliesByTopic(int topicId)

{

return \_ctx.Replies.Where(r => r.TopicId == topicId);

}

}