Доступ к данным при помощи Entity Framework

Понятие об ORM

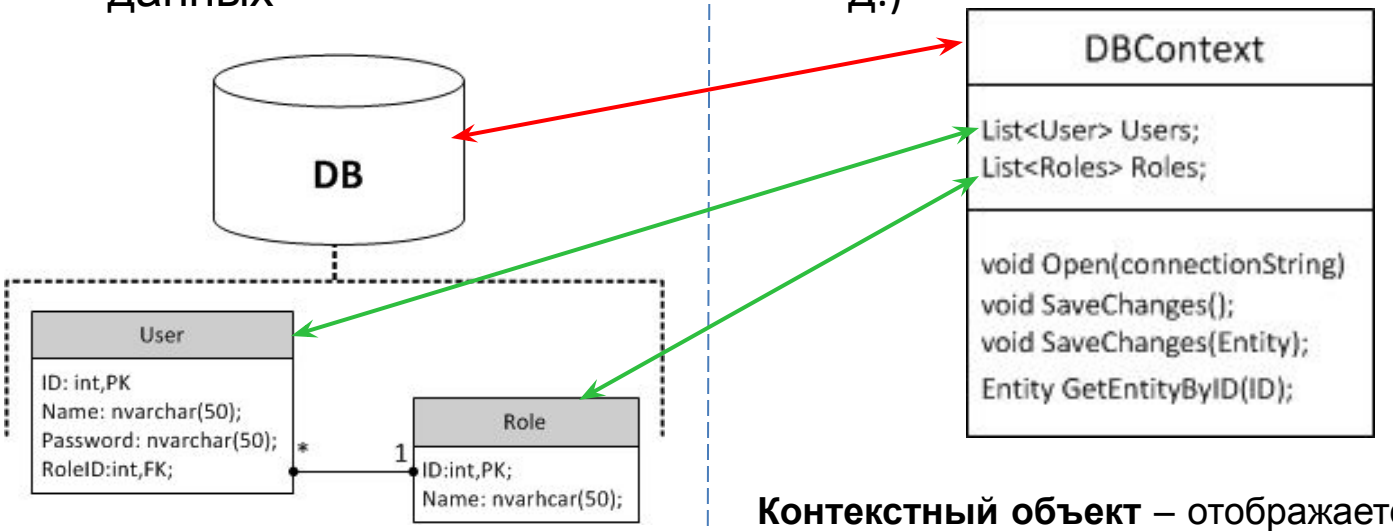
Без ORM код жестко привязан к источнику данных, программисту нужно хорошо знать **SQL.**

С ORM, вы пишете код обращения к базе на используемом языке. ORM преобразует этот код в SQL и выполняет обращение к источнику:

Как ORM выполняет отображение

Хранилище данных

Программа (C#, Java и т. д.)

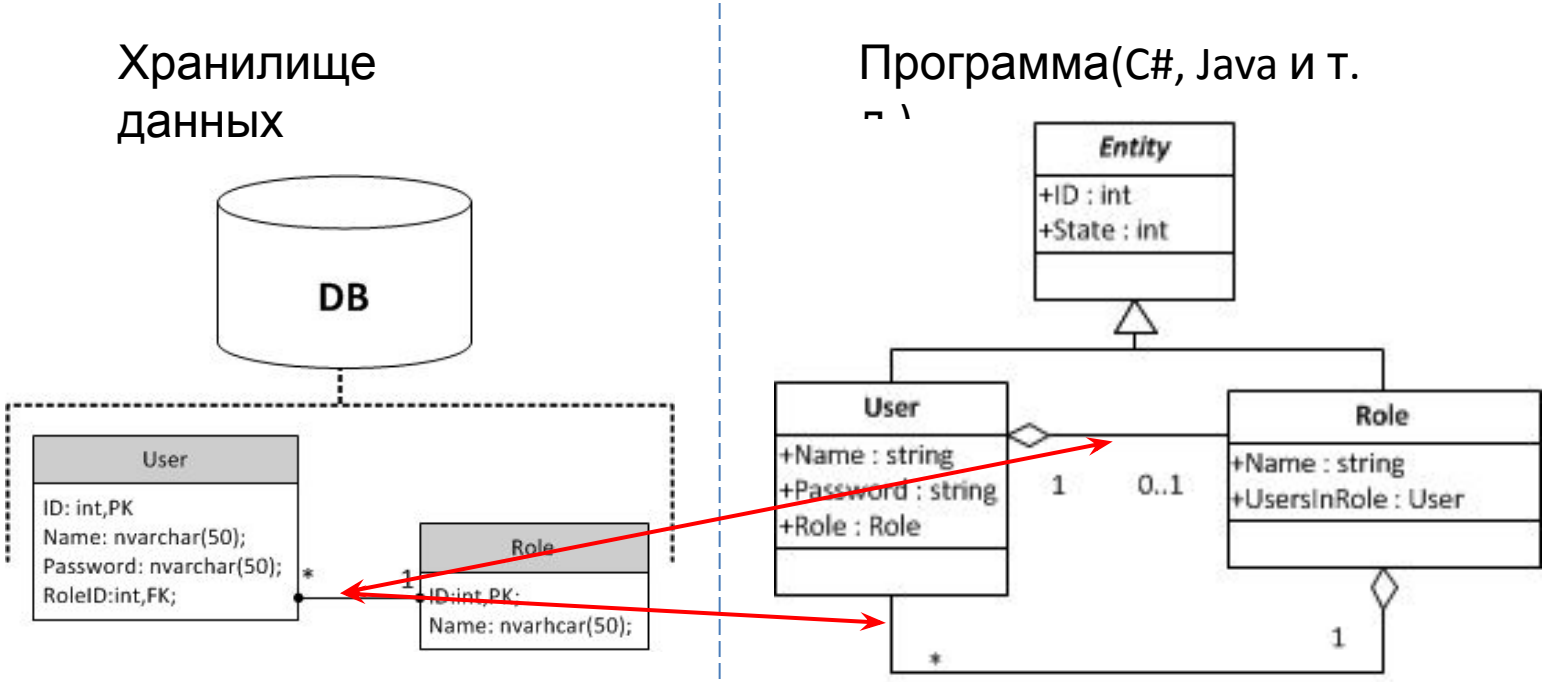


Контекстный объект – отображается на саму БД. Назначение:

- Подключение к БД;
- Выполнение запросов;
- Отслеживание изменений и выполнение сохранения данных в БД;

Коллекции – служат для отображения содержимого таблиц. При запросе данных из таблиц, требуемые объекты помещаются в соответствующие типизированные коллекции.

Как ORM выполняет отображение



Связи – отображаются на вложенные коллекции и ссылки. В данном примере:
User.Role – ссылка на объект Role, представляющий роль данного пользователя.
Role.UserInRoles – коллекция со всеми объектами пользователей, находящихся в данной роли

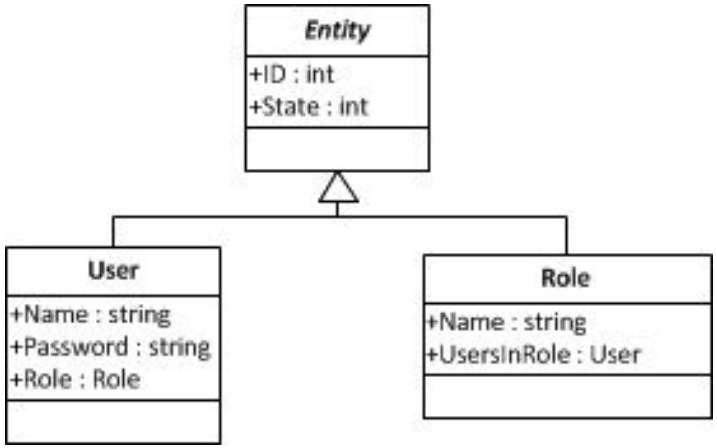
Как ORM выполняет отображение

Хранилище данных

ID	Name	Password	RoleID
1	Name1	XXXXX	1
2	Name2	XXXXX	3

ID	Name
1	Admin
2	Trusted
3	Customer

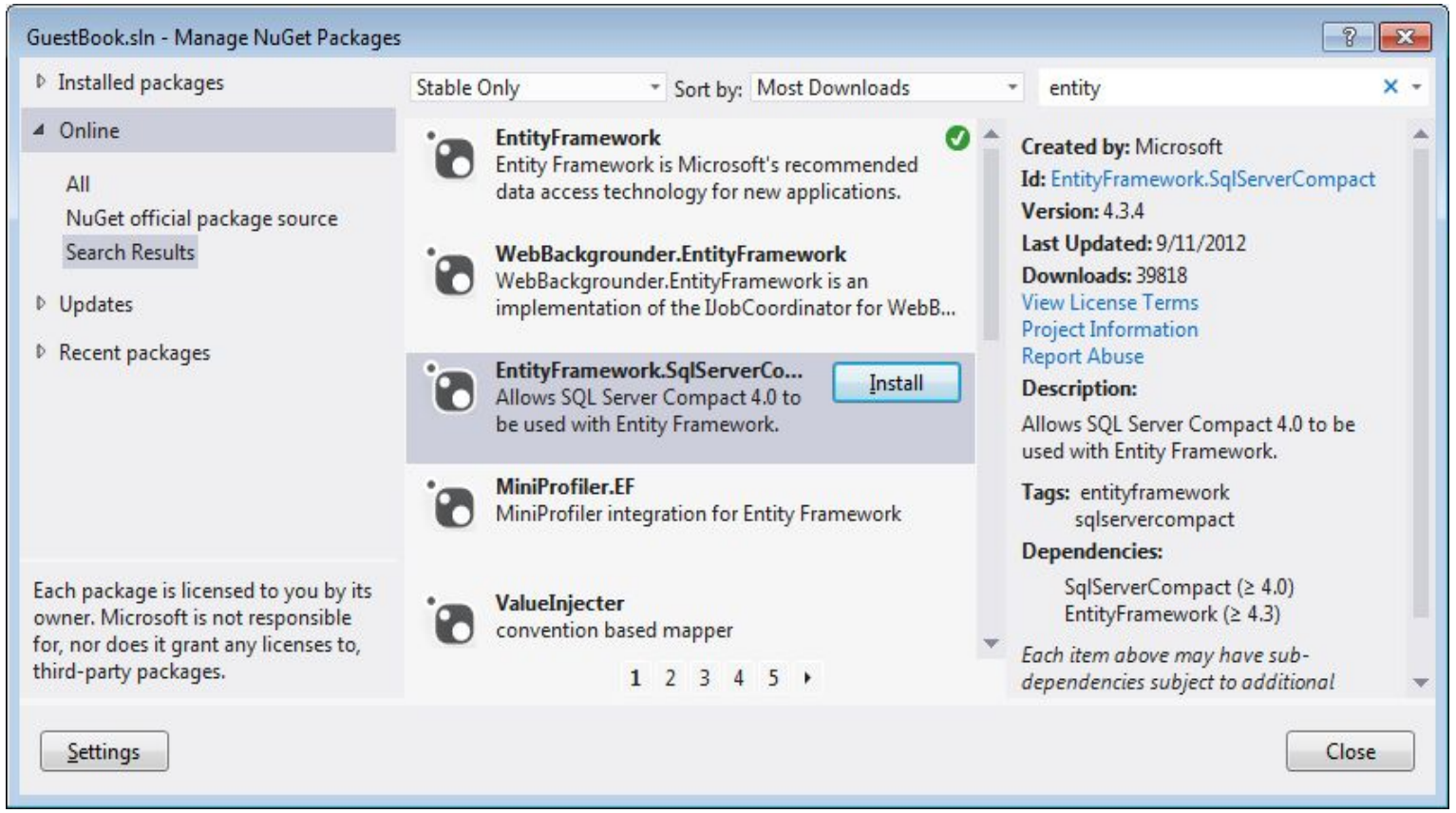
Ваше окружение (C#, Java и т. д.)



Строки таблицы отображаются на экземпляры соответствующих объектов.

Например, поле *Entity.State* позволяет отслеживать состояние объектов и осуществлять отложенную обработку этих объектов.

Entity Framework для SQL Server CE



Строительные блоки Entity Framework

Поставщик данных для EF от MS SQL Server находится в сборке System.Data.Entity.dll.
Oracle и MySQL также имеют поставщиков для EF.

Две ключевых части API-интерфейса EF — это службы объектов и клиент сущности.

Службы объектов отслеживают изменения, внесенные в сущности, управляют отношениями между сущностями, а также обеспечивают возможности сохранения изменений в базе данных. Службы работают с классами-потомками DbContext и DbSet<E>.

Клиент сущности. Пространство System. Data. EntityClient аналогично SqlClient или OdbcClient и содержит классы EntityConnection, EntityCommand и т.п. Эти классы работают за кулисами, но можно их применять непосредственно.

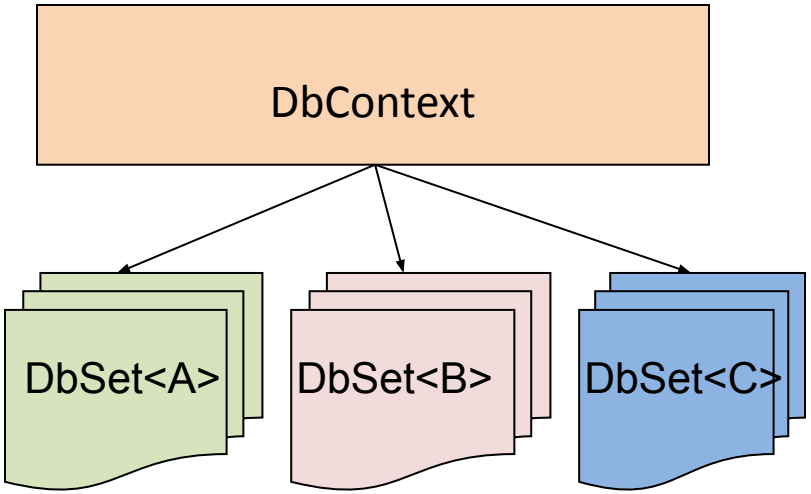
Файл *. edmx

Хотя сущности клиентской стороны в конечном итоге отображаются на таблицу базы данных, жесткая связь между ними отсутствует.

Файл *. edmx содержит XML-описания для:

1) сущностей,
2) физической базы данных,
3) инструкций по отображению сущностей на таблицы и ограничения базы данных.

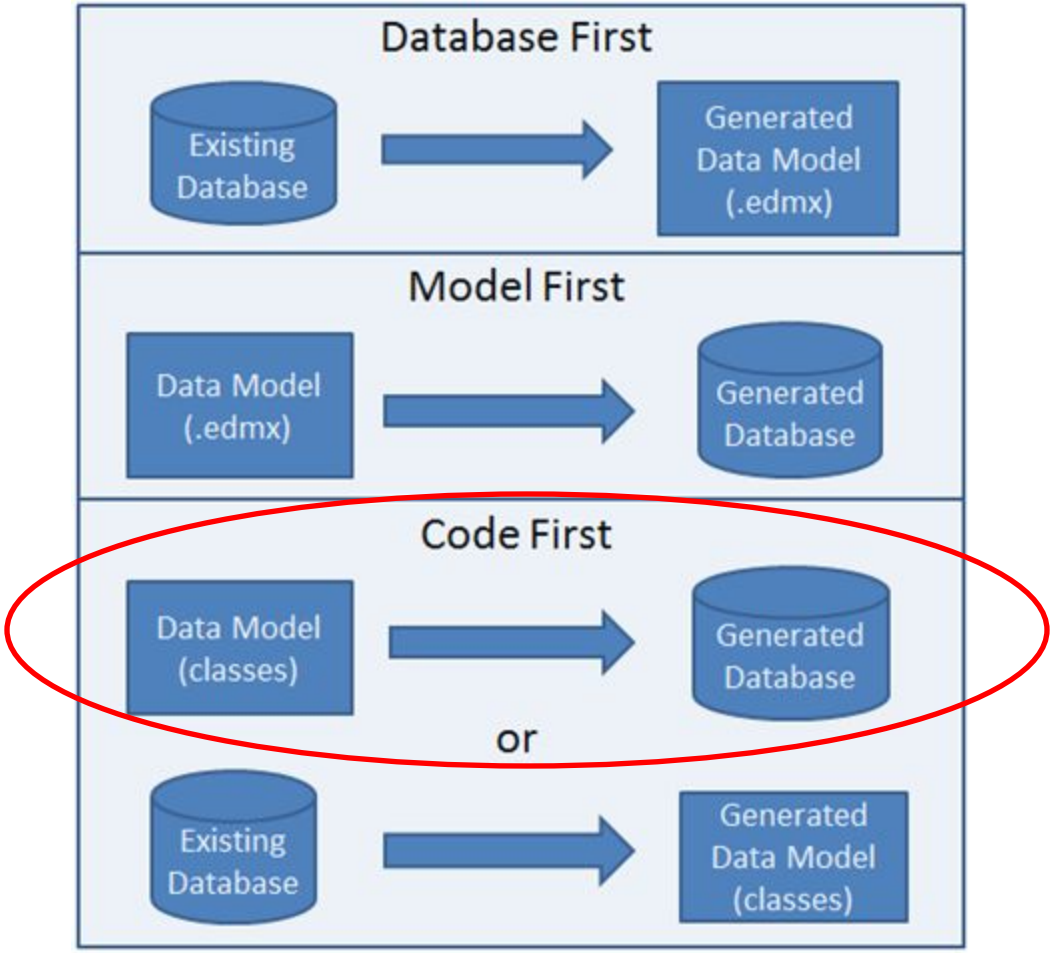
Классы DbContext и DbSet<TEntity>




```
class DbContext : IDisposable
{
    public virtual int SaveChanges();

    protected virtual void OnModelCreating(DbModelBuilder modelBuilder);
}
```

```
class DbSet<TEntity> : IEnumerable<TEntity>
{
    public TEntity Add(TEntity entity);

    public TEntity Find(params object[] keyValues);

    public TEntity Remove(TEntity entity);
}
```

Стратегии моделирования



Стратегия CodeFirst

1. Определить модели сущностей.
2. Определить модель контекста.
3. Прописать строку соединения в web.config.

Имя строки соединения должно совпадать с именем класса контекста.

Если база уже существует, таблицы в ней не будут сгенерированы.

Модели контекста и сущностей

Модель контекста должна наследовать класс DbContext.

```
using System;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace GuestBook.Models
{
   public class Repository: DbContext
   {
      public DbSet<Record> Comments { get; set; }
   }
}
```

И иметь свойства – коллекции сущностей типа DbSet<T>

```
public class Record
{
    public int Id { set; get; }
    public string Text { set; get; }
    public string Author { set; get; }
    public DateTime RecordDate { set; get; }
}
```

Строка соединения в Web.config

```
<add name= "Repository" providerName="System.Data.SqlServerCe.4.0"
        connectionString="Data Source=|DataDirectory|GBE.sdf" />
```

Перевод гостевой книги на EF

Создать слой доступа к данным по шаблону Репозиторий.

```
public class Repository : DbContext, IRepository
{
    public DbSet<Record> Records { get; set; }
    public IEnumerable<Record> Read()
    {
        return Records as IEnumerable<Record>;
    }
    public void Create(Record record)
    {
        record.RecordDate = DateTime.Now;
        Records.Add(record);
        SaveChanges();
    }
}
```

Обнаружение типов-сущностей

- Сущностями считаются базовые типы коллекций DbSet в классе контекста.
- Сущностями также считаются все типы, на которые ссылаются сущности, и это правило рекурсивно.

Сущность

Сущность

```
public class Repository: DbContext
{
     public DbSet<Record> Products { get; set; }
}

public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public Category Category { get; set; }
}

public class Category
{
   public int Id { get; set; }
   public string Name { get; set; }
   public ICollection<Product> Products { get; set; }
}
```

Чтобы исключить какой-то тип из этой цепи, его надо пометить атрибутом [NotMapped]

Интерфейс IRepository

Выделяем интерфейс IRepository:

```
using System;
namespace GB.Models
{
    interface IRepository
    {
        void Create(Record record);
        System.Collections.Generic.List<Record> Read();
    }
}
```

Класс Repository реализует IRepository:

```
public class Repository : IRepository {…}
```

Изменения в контроллере

Было:

```
public class HomeController : Controller
{
    Repository repository = new Repository();

    public ActionResult Index()
    {
        return View(repository.Read());
    }
    …
}
```

Стало:

```
public class HomeController : Controller
{
    IRepository repository;

    public HomeController(IRepository repository)
    {
        this.repository = repository;
    }

    public ActionResult Index()
    {
        return View(repository.Read());
    }
    …
 }
```

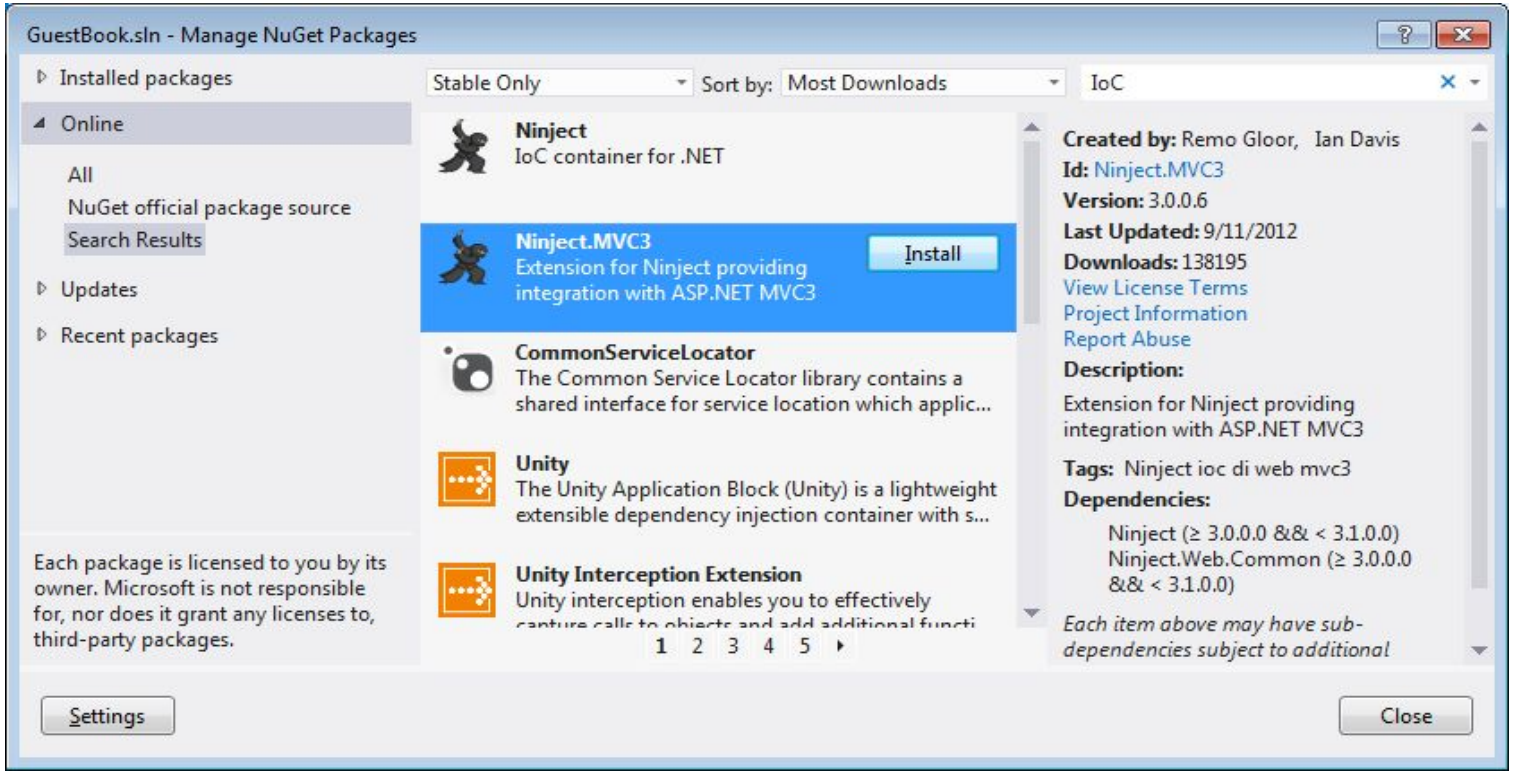
У нас проблема…

- Фабрика контроллеров из фреймворка MVC готова воспользоваться любым конструктором контроллера, но "не знает", что в него передавать.

- Выход в использовании службы, которая:
  1) знает, как реализовать интерфейс IRepository;
  2) доступна фабрике контроллеров.

  т.е. Ninject

Установка пакета Ninject.MVC3



Настройка зависимостей

В папке App_Start появился файл NinjectWebCommon.cs, который содержит весь необходимый для работы Ninject код.

Метод RegisterServices предназначен для задания правил реализации интерфейсов.

```
/// <summary>
/// Load your modules or register your services here!
/// </summary>
/// <param name="kernel">The kernel.</param>
private static void RegisterServices(IKernel kernel)
{
    kernel.Bind<IRepository>().To<Repository>();
}
```