Создание приложений для СУБД Firebird с использованием различных компонент и драйверов:

MVC.NET и Entity Framework 6

Автор: Денис Симонов
31.03.2016

Оглавление

Создание Web приложений с использованием Entity Framework

В данной статье будет описан процесс создания web приложений для СУБД Firebird с использованием Entity Framework и среды Visual Studio 2015.

В данной статье обсуждаются особенности создания именно Web приложений, базовые принципы работы с Entity Framework и Firebird описаны в предыдущей статье.

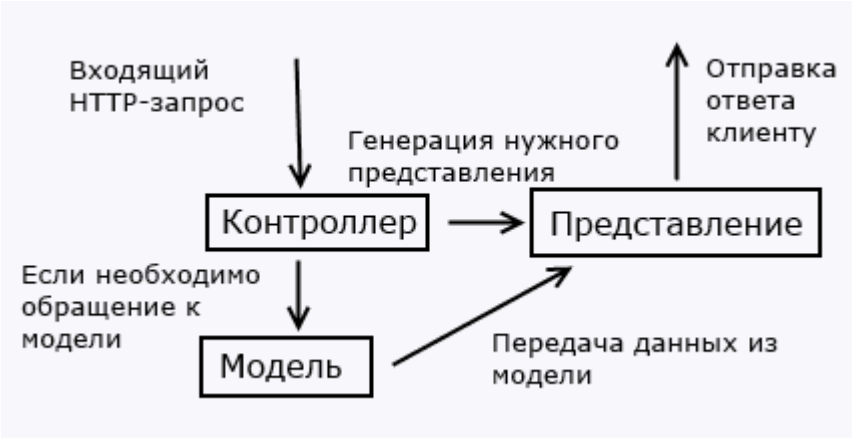
Платформа .NET предоставляет два основных фреймворка для создания web приложений: ASP.NET Web Forms и ASP.NET MVC. Я предпочитаю использовать паттерн MVC, поэтому в дальнейшем будет описываться именно эта технология.

Платформа **ASP.NET MVC** представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

- **Контроллер** (controller). Контроллеры осуществляют взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. В приложении MVC представления только отображают данные, а контроллер обрабатывает вводимые данные и отвечает на действия пользователя. Например, контроллер может обрабатывать строковые значения запроса и передавать их в модель, которая может использовать эти значения для отправки запроса в базу данных.
- **Представление** (view) — это собственно визуальная часть или пользовательский интерфейс приложения. Пользовательский интерфейс обычно создаётся на основе данных модели.
- **Модель** (model). Объекты моделей являются частями приложения, реализующими логику для работы данными приложения. Объекты моделей часто получают и сохраняют состояние модели в базе данных.

Общую схему взаимодействия этих компонентов можно представить следующим образом:



Шаблон MVC позволяет создавать приложения, различные аспекты которых (логика ввода, бизнес-логика и логика интерфейса) разделены, но достаточно тесно взаимодействуют друг с другом. Эта схема указывает расположение каждого вида логики в приложении. Пользовательский интерфейс располагается в представлении. Логика ввода располагается в контроллере. Бизнес-логика находится в модели. Это разделение позволяет работать со сложными структурами при создании приложения, так как обеспечивает одновременную реализацию только одного аспекта. Например, разработчик может сконцентрироваться на создании представления отдельно от бизнес-логики.

Более полную информацию о технологии ASP.NET MVC вы можете найти на сайте сообщества ASP.NET.

Помимо библиотек для работы с Firebird, Entity Framework и MVC.NET нам потребуется множество JavaScript библиотек для поддержки отзывчивого интерфейса, таких как jquery, jquery-ui, Bootstrap, jqGrid. В этом примере мы постарались приблизить интерфейс веб-приложения к настольным приложениям, активно применяя гриды для отображения и модальные окна для ввода данных.

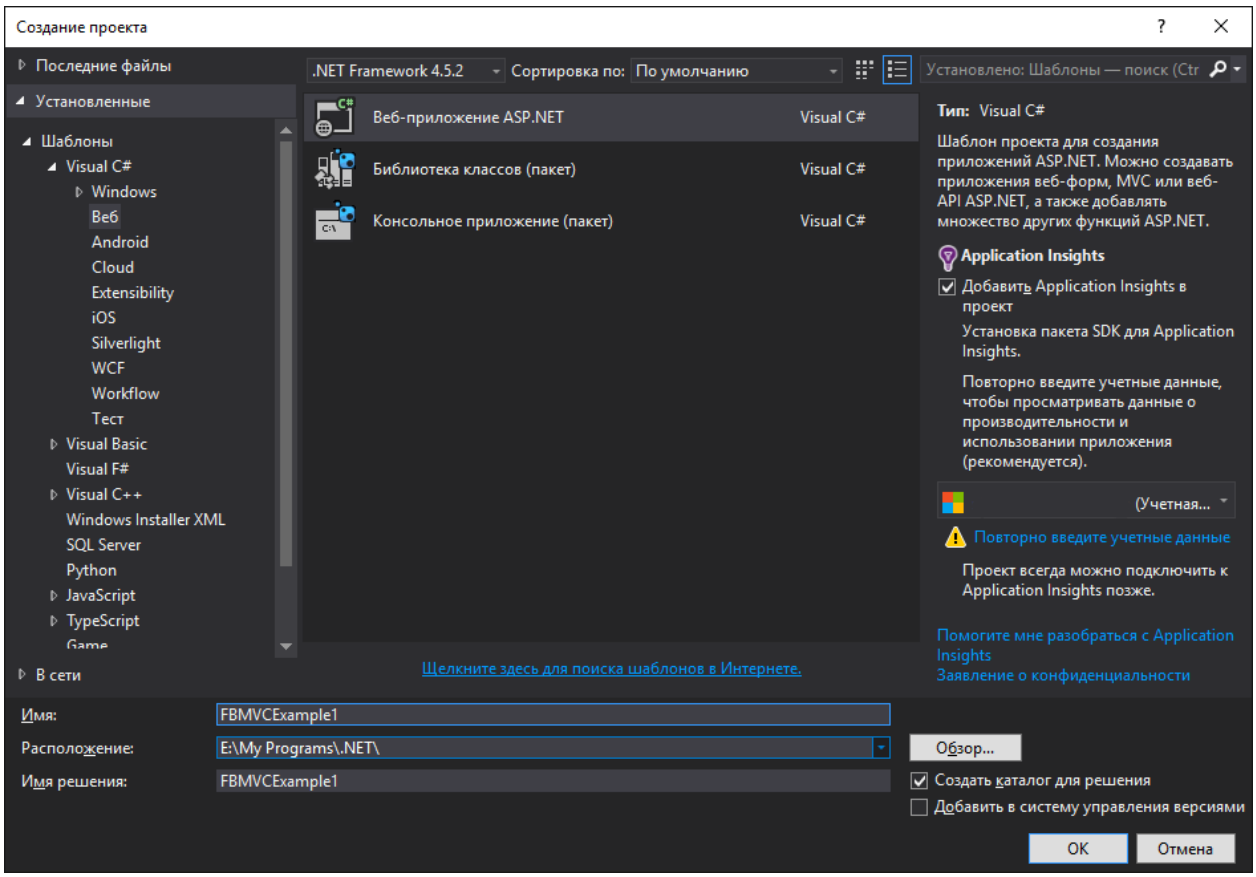
Подготовка Visual Studio 2015 для работы с Firebird

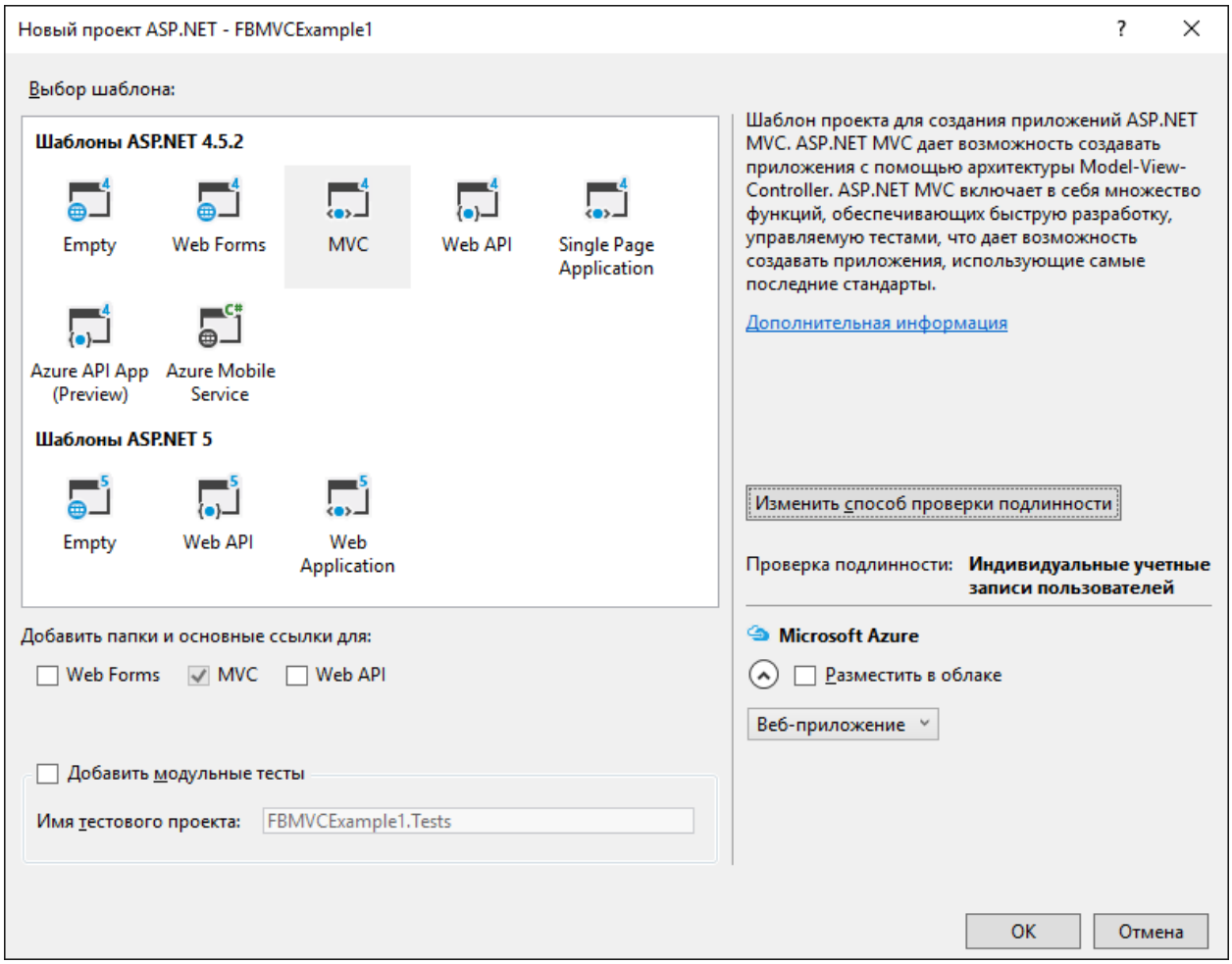
Для работы Visual Studio с СУБД Firebird вам придётся проделать несколько дополнительных шагов, которые подробно были описаны в предыдущей статье.

Создание проекта

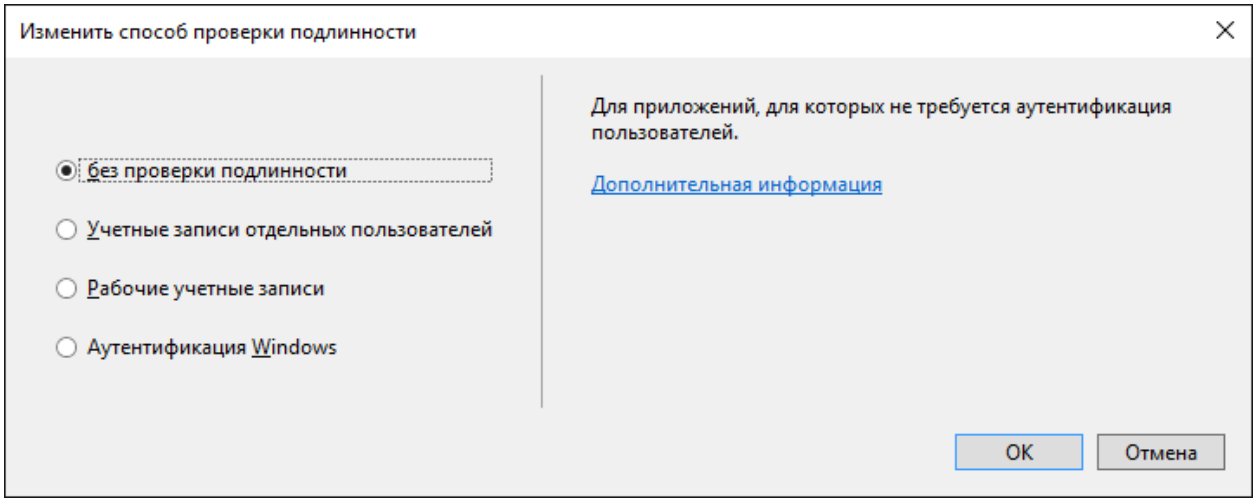
В данной главе мы покажем, как создаётся каркас MVC.NET приложения с помощью мастеров Visual Studio.

Итак, откроем Visual Studio 2015 Файл -> Создать -> Проект и создадим новый проект. Назовём новый проект FBMVCExample.


Создание проекта
Последние файлы
Установленные
Шаблоны
Visual C#
Windows
Веб
Android
Cloud
Extensibility
iOS
Silverlight
WCF
Workflow
Тест
Visual Basic
Visual F#
Visual C++
Windows Installer XML
SQL Server
Python
JavaScript
TypeScript
В сети
.NET Framework 4.5.2
Сортировка по:
По умолчанию
Установлено: Шаблоны — поиск (Ctr
Веб-приложение ASP.NET
Visual C#
Библиотека классов (пакет)
Visual C#
Консольное приложение (пакет)
Visual C#
Щелкните здесь для поиска шаблонов в Интернете.
Тип: Visual C#
Шаблон проекта для создания приложений ASP.NET. Можно создавать приложения веб-форм, MVC или веб-API ASP.NET, а также добавлять множество других функций ASP.NET.
Application Insights
Добавить Application Insights в проект
Установка пакета SDK для Application Insights.
Повторно введите учетные данные, чтобы просматривать данные о производительности и использовании приложения (рекомендуется).
(Учетная...
Повторно введите учетные данные
Проект всегда можно подключить к Application Insights позже.
Помогите мне разобраться с Application Insights
Заявление о конфиденциальности
Имя:
FBMVCExample1
Расположение:
E:\My Programs\.NET\
Имя решения:
FBMVCExample1
Обзор...
Создать каталог для решения
Добавить в систему управления версиями
OK
Отмена


Новый проект ASP.NET - FBMVCExample1
Выбор шаблона:
Шаблоны ASP.NET 4.5.2
Empty
Web Forms
MVC
Web API
Single Page Application
Azure API App (Preview)
Azure Mobile Service
Шаблоны ASP.NET 5
Empty
Web API
Web Application
Шаблон проекта для создания приложений ASP.NET MVC. ASP.NET MVC дает возможность создавать приложения с помощью архитектуры Model-View-Controller. ASP.NET MVC включает в себя множество функций, обеспечивающих быструю разработку, управляемую тестами, что дает возможность создавать приложения, использующие самые последние стандарты.
Дополнительная информация
Изменить способ проверки подлинности
Проверка подлинности: Индивидуальные учетные записи пользователей
Microsoft Azure
Разместить в облаке
Веб-приложение
Добавить папки и основные ссылки для:
Web Forms
MVC
Web API
Добавить модульные тесты
Имя тестового проекта:
FBMVCExample1.Tests
OK
Отмена

Изменим способ проверки подлинности. В данный момент создадим веб приложение без проверки подлинности. К этому вопросу мы вернёмся чуть позже.



После этого будет создан проект, который практически не обладает никакой функциональностью, хотя уже имеет базовую структуру.

Далее следует краткое описание этой структуры.

Папка или файл	Описание
/App_Data	В эту папку помещаются закрытые данные веб приложения, такие как XML-файлы или файлы базы данных.
/App_Start	Эта папка содержит ряд основных настроек конфигурации для проекта, в том числе определение маршрутов и фильтров.
/Content	Сюда помещается статическое содержимое, такое как CSS-файлы и изображения. Это является необязательным соглашением. Вы можете хранить файлы стилей в любом подходящем месте.
/Controllers	Сюда помещаются классы контроллеров. Это необязательное соглашение. Вы можете классы контроллеров где угодно.
/Models	Сюда помещаются классы моделей представлений и моделей предметной области, хотя все кроме простейших приложений выигрывают от определения модели предметной области в отдельном проекте. Это необязательное соглашение. Вы можете размещать классы моделей в любом удобном месте.
/Scripts	Эта папка предназначена для хранения библиотек JavaScript, используемых в приложении. По умолчанию Visual Studio добавляет библиотеки jQuery и несколько других популярных JavaScript-библиотек. Это необязательное соглашение.
/Views	В этой папке хранятся представления и частичные представления, обычно сгруппированные вместе в папках с именами контроллеров, с которыми они связаны.

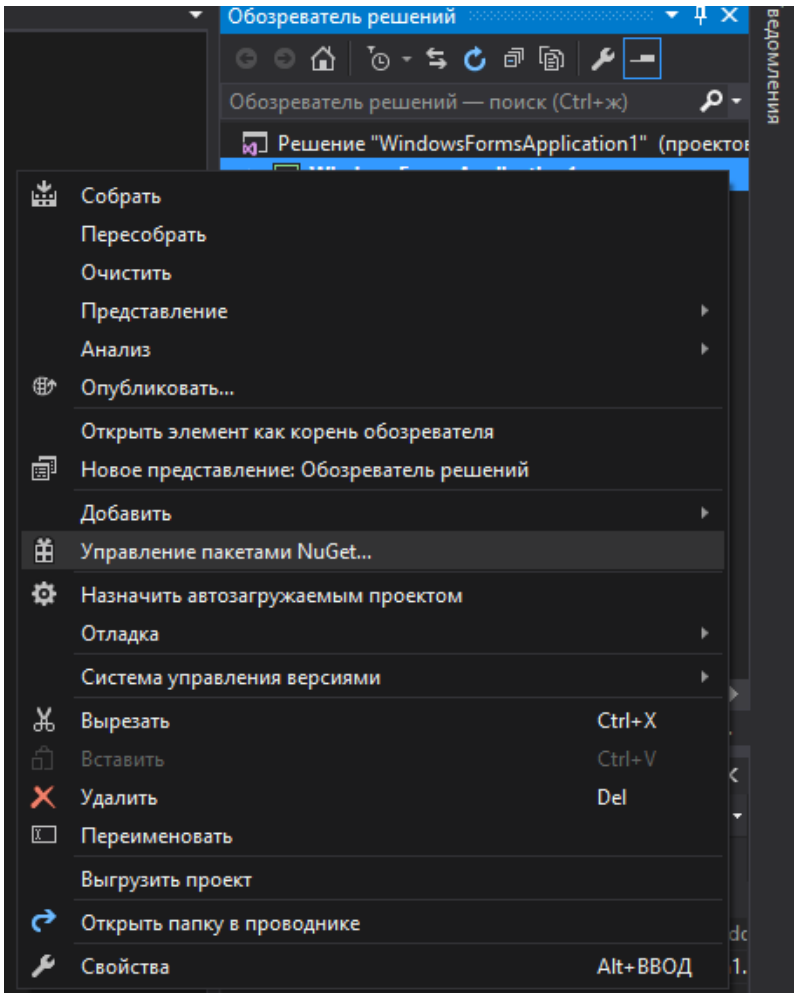
/Views/Shared	В этой папке хранятся компоновки и представления, не являющиеся специфичными для какого-либо контроллера.
/Views/Web.config	Это конфигурационный файл. В нем содержится конфигурационная информация, которая обеспечивает обработку представлений с помощью ASP.NET и предотвращает их обслуживание веб-сервером IIS, а также пространства имён, по умолчанию импортируемые в представления.
/Global.asax	Это глобальный класс приложения ASP.NET. В файле его кода (Global.asax.cs) регистрируется конфигурация маршрутов, а также предоставляется любой код, который должен выполняться при запуске или завершении приложения либо в случае возникновения необработанного исключения.
/Web.config	Конфигурационный файл для приложения.

Теперь добавим необходимые пакеты с помощью менеджера пакетов NuGet. Нам потребуются установить недостающие пакеты:

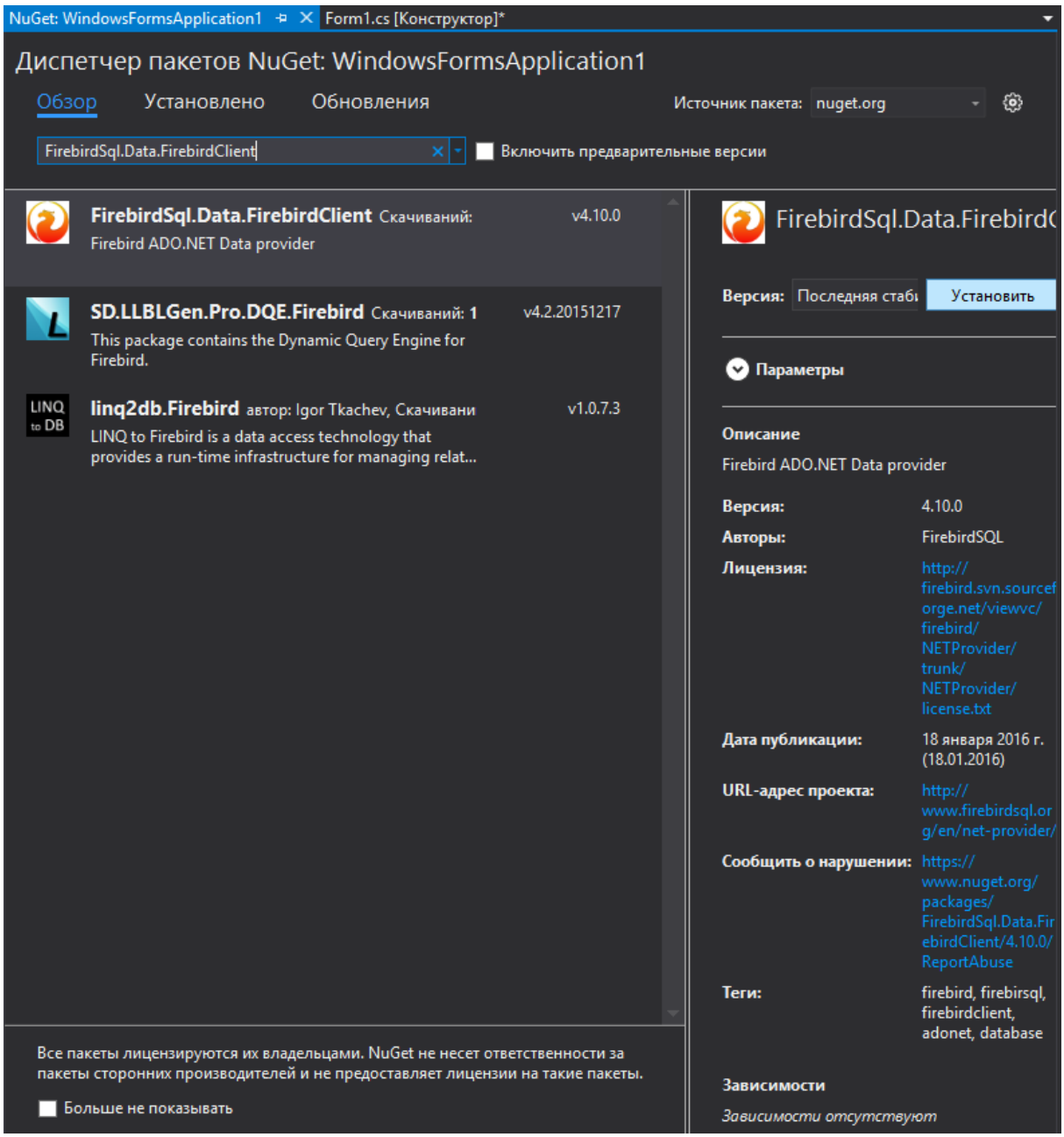
- FirebirdSql.Data.FirebirdClient
- EntityFramework (автоматически добавлен мастером)
- EntityFramework.Firebird
- Bootstrap (автоматически добавлен мастером)
- jQuery (автоматически добавлен мастером)
- jQuery.UI.Combined
- Respond (автоматически добавлен мастером)
- Newtonsoft.Json
- Moderninzr (автоматически добавлен мастером)
- Trirand.jqGrid

Замечание
Не все пакеты, предоставляемые NuGet, являются библиотеками последних версий. Особенно это касается JavaScript библиотек. Вы можете подключать последние версии JavaScript библиотек, используя CDN или просто скачать их, заменив библиотеки, предоставленные NuGet.

Для этого необходимо щёлкнуть правой клавишей мыши по имени проекта в обозревателе решений и в выпадающем меню выбрать пункт «Управление пакетами NuGet».



В появившемся менеджере пакетов произвести поиск и установку необходимых пакетов.

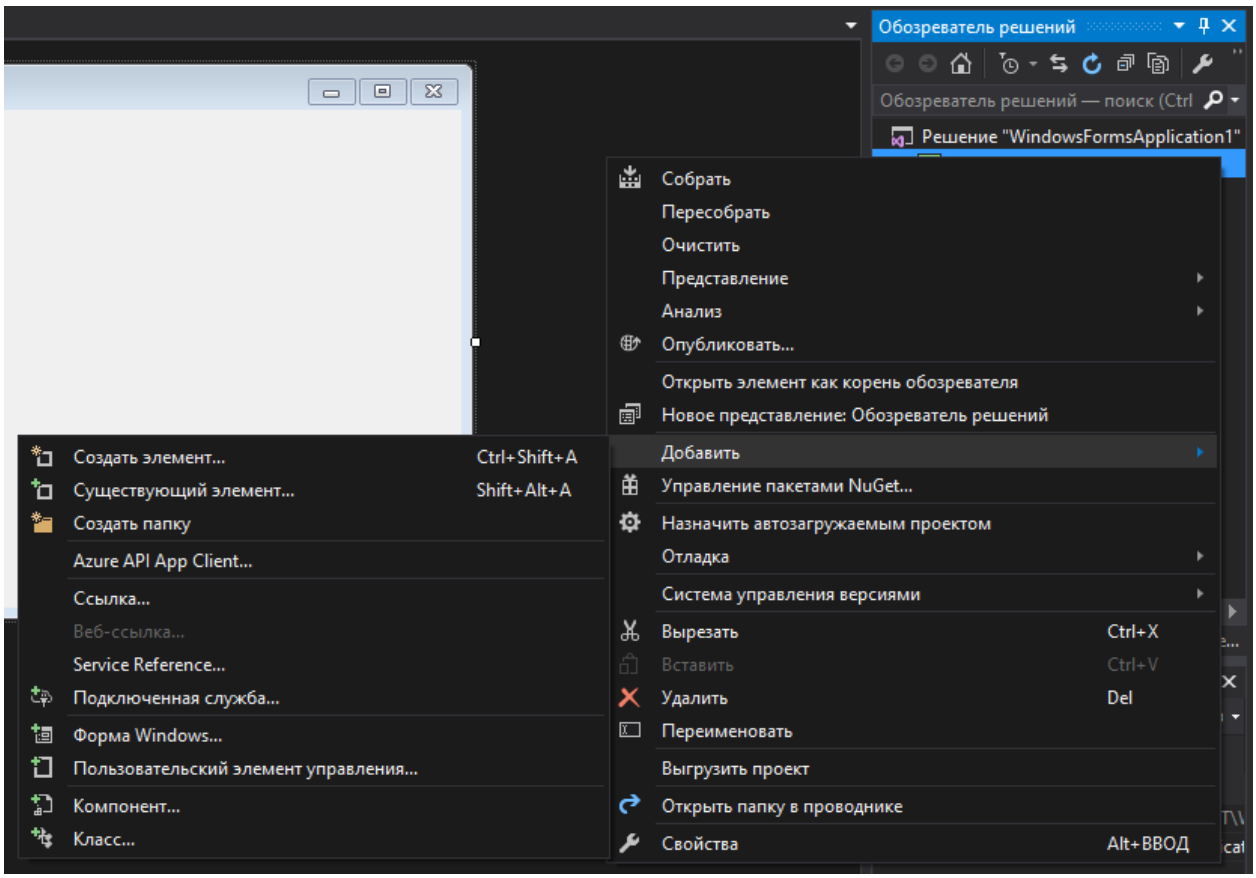


Создание EDM модели

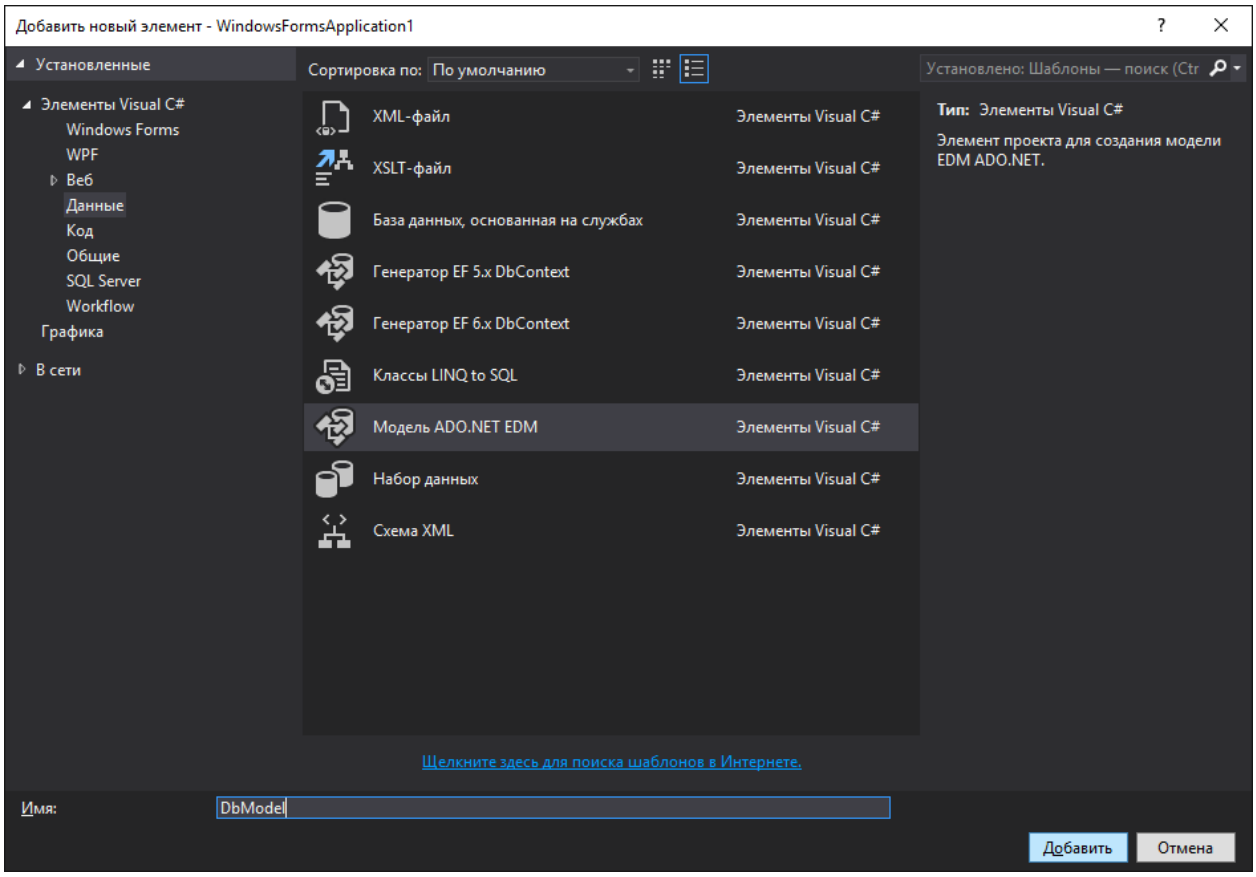
Если у вас уже есть Windows Forms приложение, которое использует Entity Framework, то вы просто можете перенести кассы моделей в папку Models. В противном случае вам необходимо будет создать их с нуля.

В своём приложении мы будем использовать подход Code First.

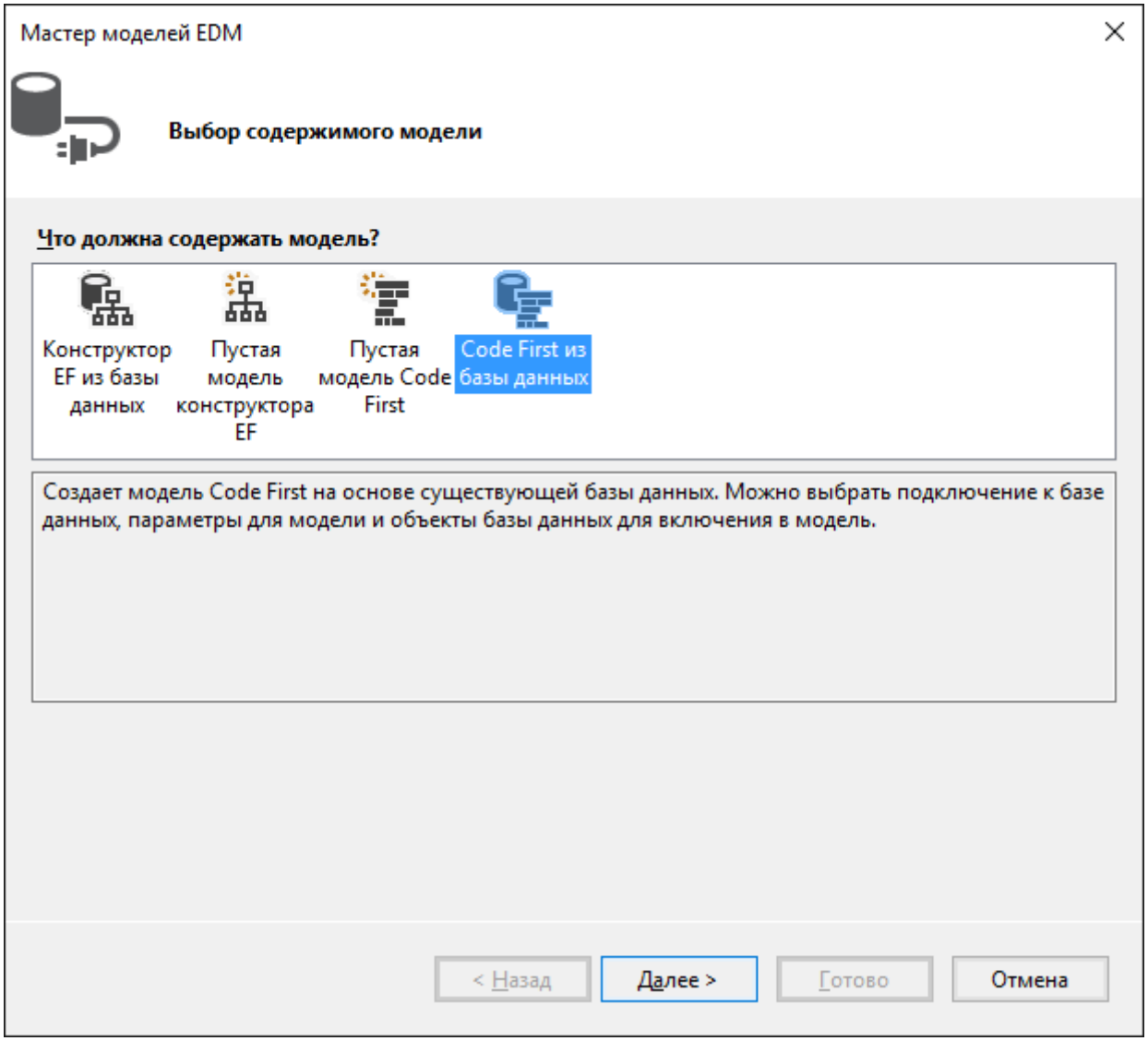
Для создания модели EDM необходимо щёлкнуть правой клавишей мыши по имени проекта в обозревателе решений и выбрать пункт меню Добавить -> Создать элемент.



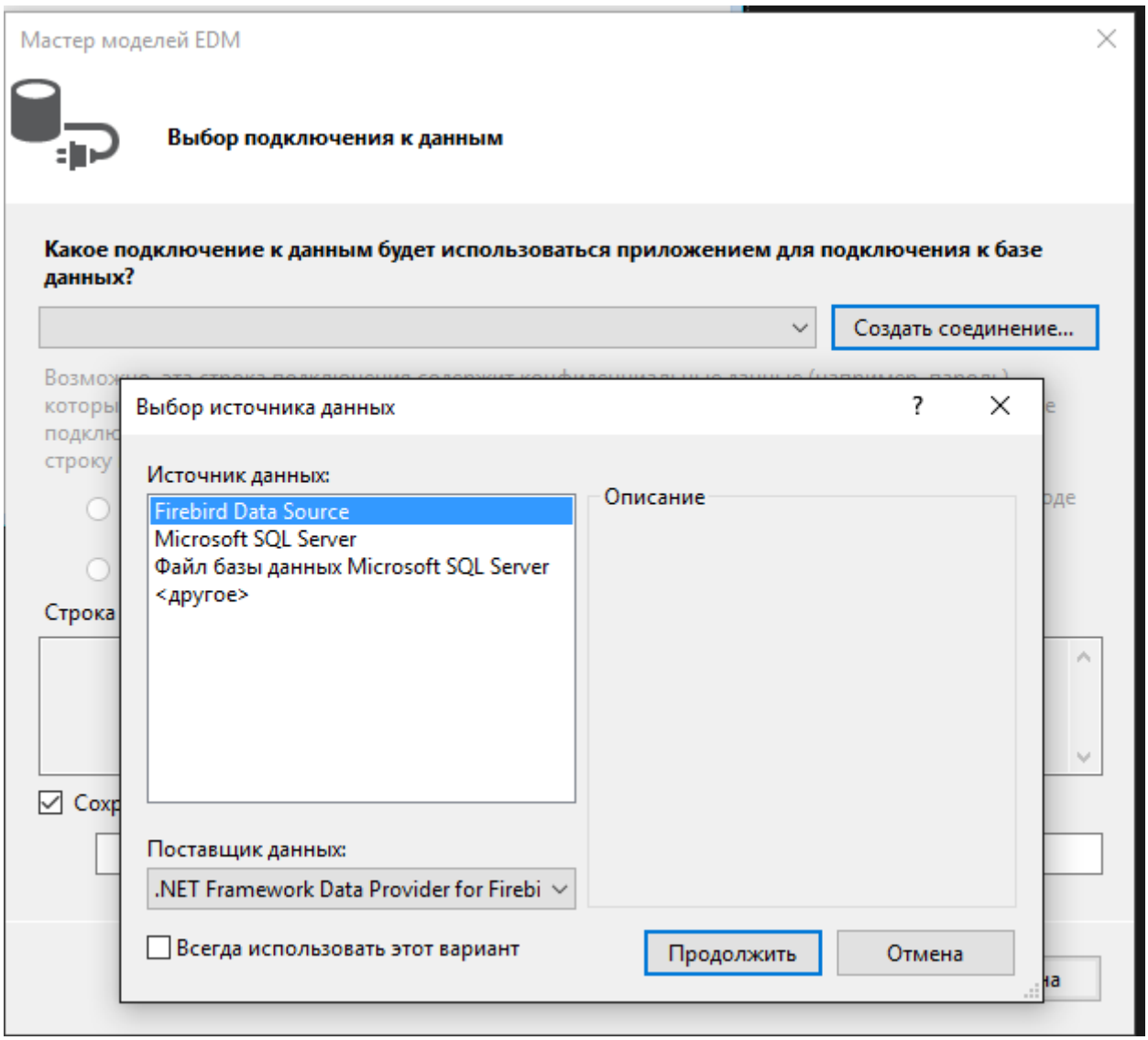
Далее в мастере добавления нового элемента выбираем пункт «Модель ADO.NET EDM».

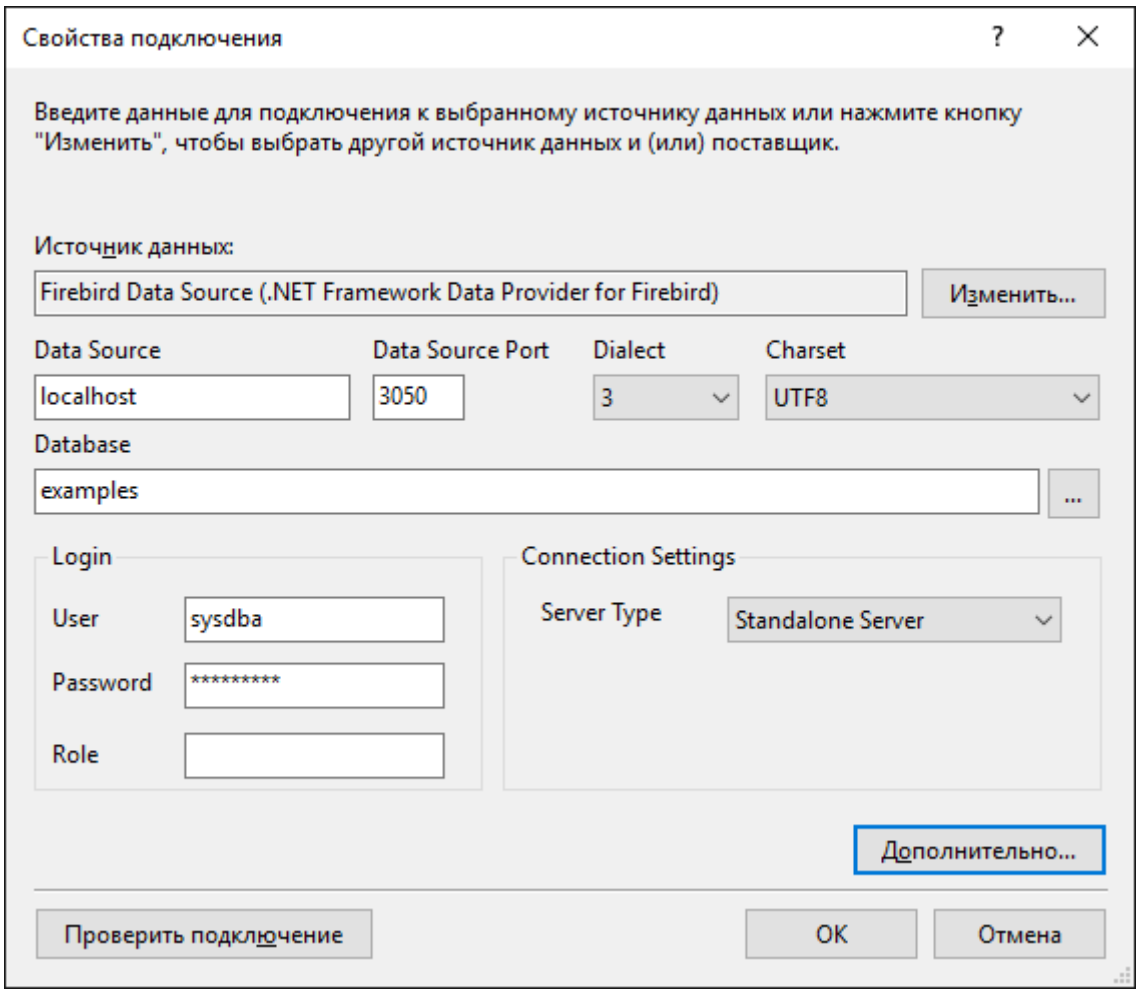


Поскольку у нас уже существует база данных (см. предыдущую статью), то будем генерировать EDM модель из базы данных.

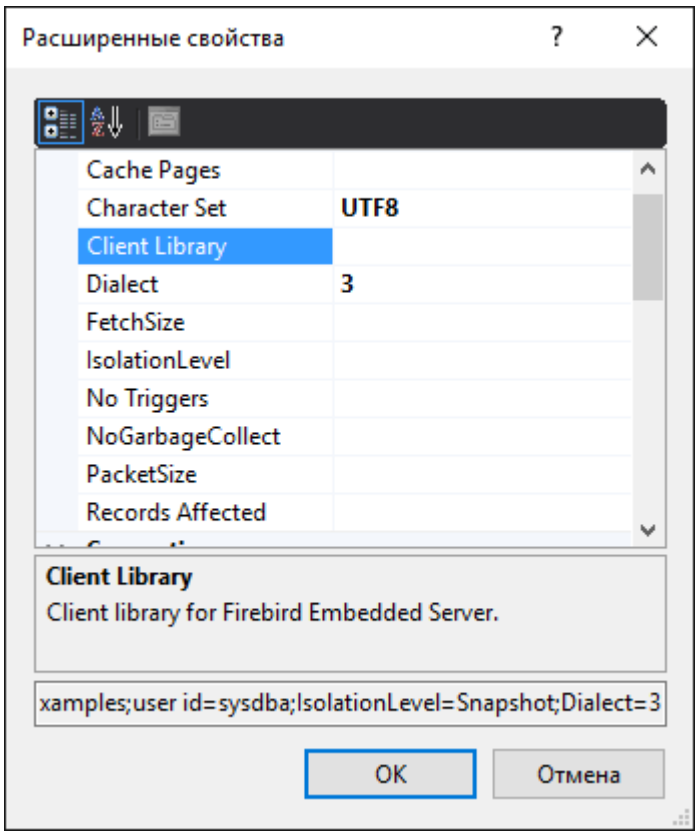


Теперь надо выбрать подключение, из которого будет создана модель. Если Такого подключения нет, то его надо создать.

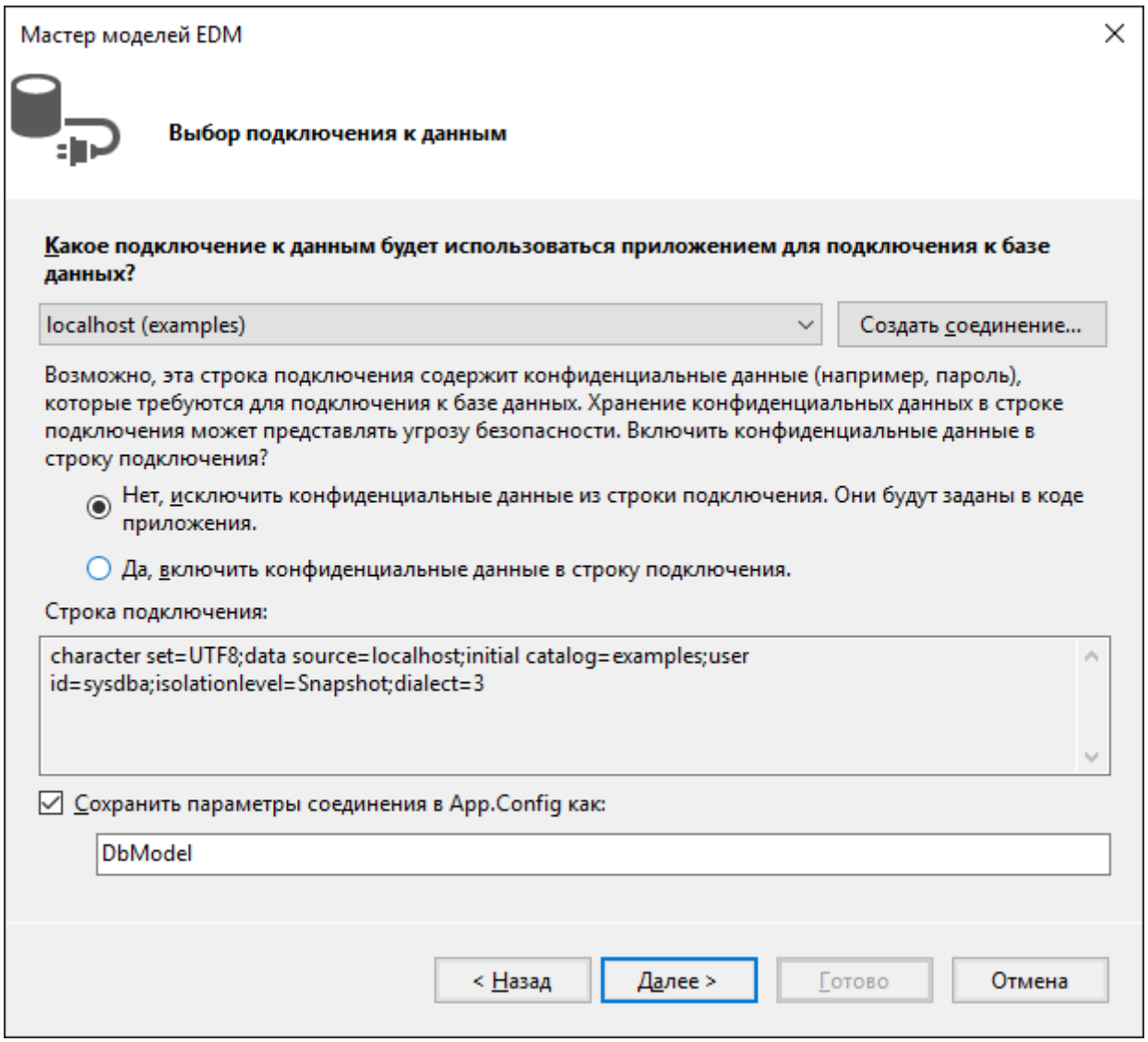

Мастер моделей EDM
Выбор подключения к данным
Какое подключение к данным будет использоваться приложением для подключения к базе данных?
Создать соединение...
Выбор источника данных
Источник данных:
Firebird Data Source
Microsoft SQL Server
Файл базы данных Microsoft SQL Server
<другое>
Описание
Поставщик данных:
.NET Framework Data Provider for Firebi
Всегда использовать этот вариант
Продолжить
Отмена



Кроме основных параметров подключения могут потребоваться также указать ряд дополнительных параметров, например, уровень изолированности транзакций (по умолчанию Read Commited), использование пула подключений и т.д. Поскольку Entity Framework (как впрочем, ADO.NET в целом) использует отсоединённую модель взаимодействия, при которой каждое подключение и транзакция активна очень короткий промежуток времени, то я бы рекомендовал задать режим изолированности Snapshot.



В процессе работы мастера создания модели у вас спросят, как хранить строку подключения.



Поскольку мы создаём веб приложение, где все пользователи будут работать с базой данных под одной и той же учётной записью, то смело выбираем «Да». В качестве имени пользователя может быть указан любой пользователь с достаточными привилегиями. Желательно не использовать пользователя SYSDBA, поскольку он обладает повышенными привилегиями, которые не требуются для функционирования веб приложения. Вы всегда можете это изменить в готовом приложении, просто отредактировав строку подключения в файле конфигурации приложения <AppName>.exe.conf. Строка подключения будет сохранена в секции **connectionStrings** примерно в таком виде

```
<add name="DbModel" connectionString="character set=UTF8; data source=localhost;initial
catalog=examples; port number=3050; user id=sysdba; dialect=3; isolationlevel=Snapshot;
pooling=True; password=masterkey;" providerName="FirebirdSql.Data.FirebirdClient" />
```

Замечание о работе с Firebird 3.0

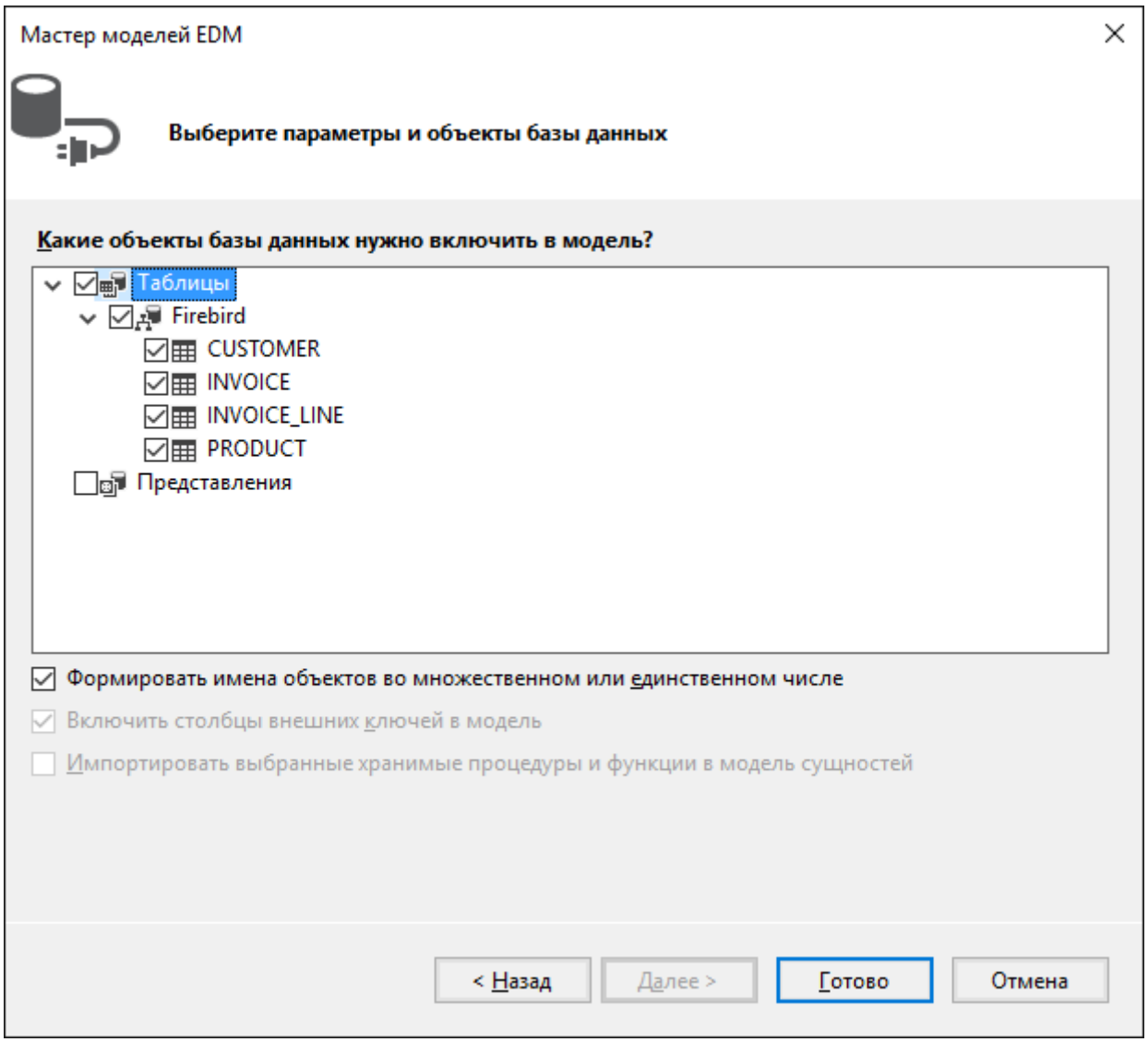
Если вы используете ADO .Net провайдер для Firebird версии ниже чем 5.0.0, то нужно учитывать, что он не поддерживает аутентификацию по протоколу SRP (по умолчанию в Firebird 3.0). Поэтому если вы желаете

работать с Firebird 3.0 со старым ADO.NET провайдером, то вам необходимо изменить некоторые настройки в firebird.conf (или в databases.conf для конкретной БД), чтобы Firebird работал через Legacy_Auth. Для этого необходимо поменять следующие настройки:

UserManager = Legacy_UserManager
WireCrypt = Disabled
AuthServer = Legacy_Auth, Srp, WinSspi

Сохранить настройки. После чего необходимо создать пользователя SYSDBA и других пользователей с использованием Legacy_UserManager.

Далее у вас спросят, какие таблицы и представления должны быть включены модель.



В принципе EDM модель готова. После работы этого мастера у вас должно появиться 5 новых файлов. Один файл модели и четыре файла описывающих каждую из сущностей модели.

Давайте посмотрим один из сгенерированных файлов описывающих сущность INVOICE.

```
    [Table("Firebird.INVOICE")]
    public partial class INVOICE
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
        public INVOICE()
        {
            INVOICE_LINES = new HashSet<INVOICE_LINE>();
        }

        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int INVOICE_ID { get; set; }

        public int CUSTOMER_ID { get; set; }

        public DateTime? INVOICE_DATE { get; set; }

        public double? TOTAL_SALE { get; set; }

        public short PAYED { get; set; }

        public virtual CUSTOMER CUSTOMER { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<INVOICE_LINE> INVOICE_LINES { get; set; }
    }
```

Класс содержат свойства, которые отображают поля таблицы INVOICE. Каждое из таких свойств снабжено атрибутами, описывающими ограничения. Подробнее об различных атрибутах вы можете почитать в документации Майкрософт Code First Data Annotations.

Кроме того, было сгенерировано ещё два навигационных свойства CUSTOMER и INVOICE_LINES. Первое содержит ссылку на сущность поставщика, второе – коллекцию строк накладных. Оно было сгенерировано потому, что таблица INVOICE_LINE имеет внешний ключ на таблицу INVOICE. Конечно, вы можете удалить это свойство из сущности INVOICE, но делать это вовсе не обязательно. Дело в том, что в данном случае свойства CUSTOMER и INVOICE_LINES использует так называемую «ленивую загрузку». При таком загрузка осуществляется при первом обращении к объекту, т.е. если связанные данные не нужны, то они не подгружаются. Однако при первом же обращении к навигационному свойству эти данные автоматически подгружаются из БД.

При использовании ленивой загрузки надо иметь в виду некоторые моменты при объявлении классов. Так, классы, использующие ленивую загрузку должны быть публичными, а их свойства должны иметь модификаторы **public** и **virtual**.

В этом же классе нас ожидает первый неприятный сюрприз. Поле TOTAL_SALE было отображено в сущности как double, хотя в базе данных оно имеет тип NUMERIC(15, 2), таким образом, мы имеем потерю точности. Я склонен расценивать это как баг в Firebird ADO.NET Provider. Давайте попробуем

исправить эту досадную оплошность. В C# существует тип decimal для операций над числами с фиксированной точностью.

```
public decimal TOTAL_SALE { get; set; }
```

Кроме того, изменим описание всех полей во всех сущностях, где используется тип Firebird NUMERIC(x, y). А именно PRODUCT.PRICE, INVOICE_LINE.QUANTITY, INVOICE_LINE.SALE_PRICE.

Теперь откроем файл DbModel.cs описывающий модель в целом.

```
    public partial class DbModel : DbContext
    {
        public DbModel()
            : base("name=DbModel")
        {
        }

        public virtual DbSet<CUSTOMER> CUSTOMERS { get; set; }
        public virtual DbSet<INVOICE> INVOICES { get; set; }
        public virtual DbSet<INVOICE_LINE> INVOICE_LINES { get; set; }
        public virtual DbSet<PRODUCT> PRODUCTS { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<CUSTOMER>()
                .Property(e => e.ZIPCODE)
                .IsFixedLength();

            modelBuilder.Entity<CUSTOMER>()
                .HasMany(e => e.INVOICES)
                .WithRequired(e => e.CUSTOMER)
                .WillCascadeOnDelete(false);

            modelBuilder.Entity<PRODUCT>()
                .HasMany(e => e.INVOICE_LINES)
                .WithRequired(e => e.PRODUCT)
                .WillCascadeOnDelete(false);

            modelBuilder.Entity<INVOICE>()
                .HasMany(e => e.INVOICE_LINES)
                .WithRequired(e => e.INVOICE)
                .WillCascadeOnDelete(false);

        }
    }
```

Здесь мы видим свойства описывающие набор данных для каждой сущности. А так же задание дополнительных свойств создания модели с помощью Fluent API. Полное описание Fluent API вы может прочитать в документации Microsoft Configuring/Mapping Properties and Types with the Fluent API.

Зададим в методе OnModelCreating точность для свойств типа decimal с помощью Fluent API. Для этого допишем следующие строчки

```
            modelBuilder.Entity<PRODUCT>()
                .Property(p => p.PRICE)
                .HasPrecision(15, 2);

            modelBuilder.Entity<INVOICE>()
```

```
            .Property(p => p.TOTAL_SALE)
            .HasPrecision(15, 2);

        modelBuilder.Entity<INVOICE_LINE>()
            .Property(p => p.SALE_PRICE)
            .HasPrecision(15, 2);

        modelBuilder.Entity<INVOICE_LINE>()
            .Property(p => p.QUANTITY)
            .HasPrecision(15, 0);
```

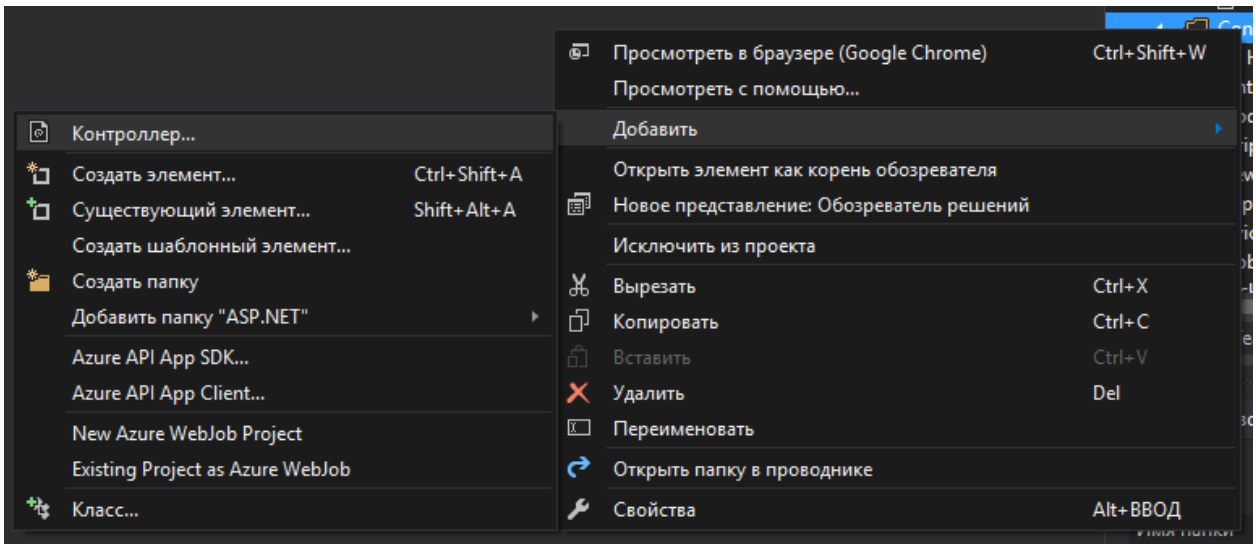
По умолчанию классы Entity Framework не предоставляют методов для удобного получения следующих значений последовательностей. Исправим это, для чего напишем следующее расширение.

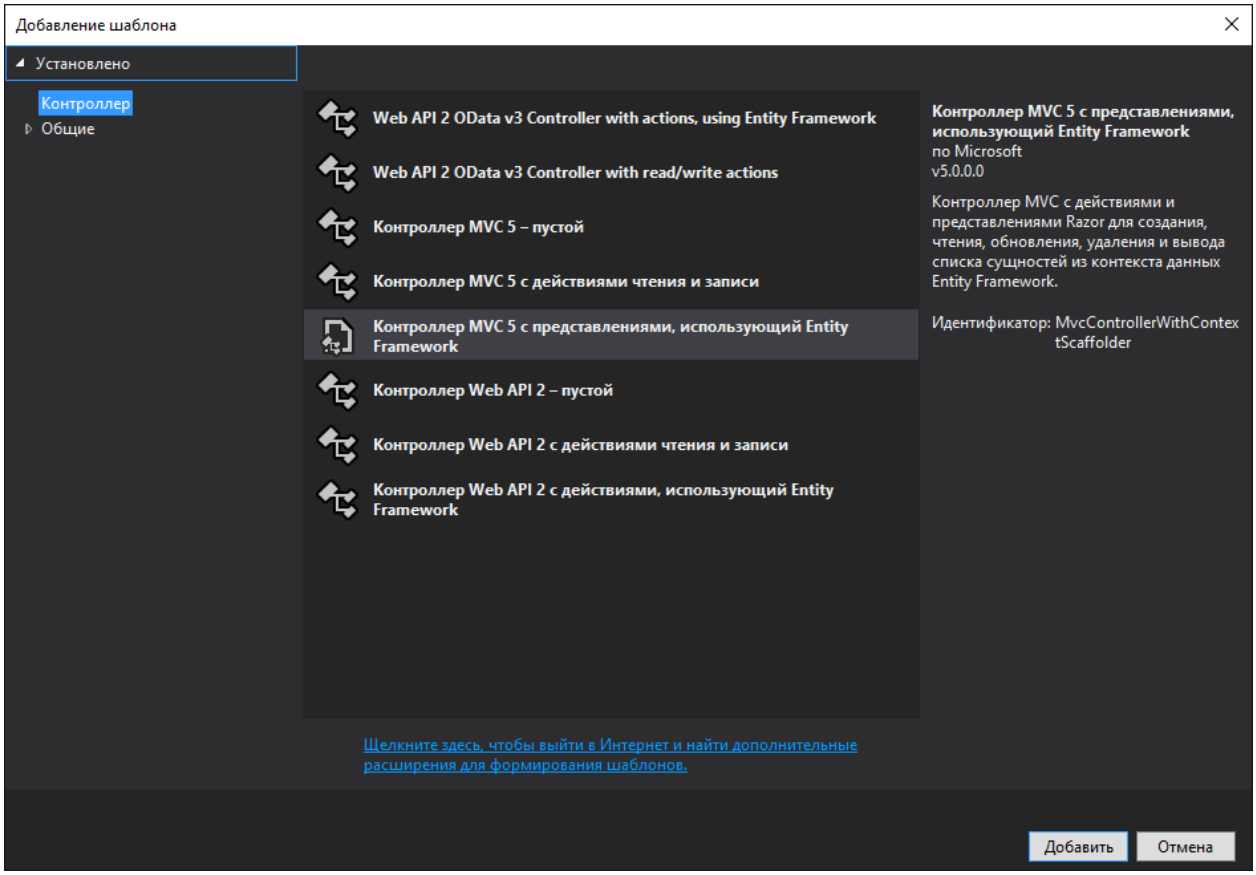
```
public static class DbExtensions
{
    /// <summary>
    /// Внутренний класс для маппинга на него значения генератора
    /// </summary>
    private class IdResult
    {
        public int Id { get; set; }
    }

    /// <summary>
    /// Получение следующего значения последовательности
    /// </summary>
    /// <param name="dbContext"></param>
    /// <param name="genName"></param>
    /// <returns>Значение последовательности</returns>
    public static int NextValueFor(this DbModel dbContext, string genName)
    {
        string sql = String.Format("SELECT NEXT VALUE FOR {0} AS Id FROM RDB$DATABASE",
genName);
        return dbContext.Database.SqlQuery<IdResult>(sql).First().Id;
    }
}
```

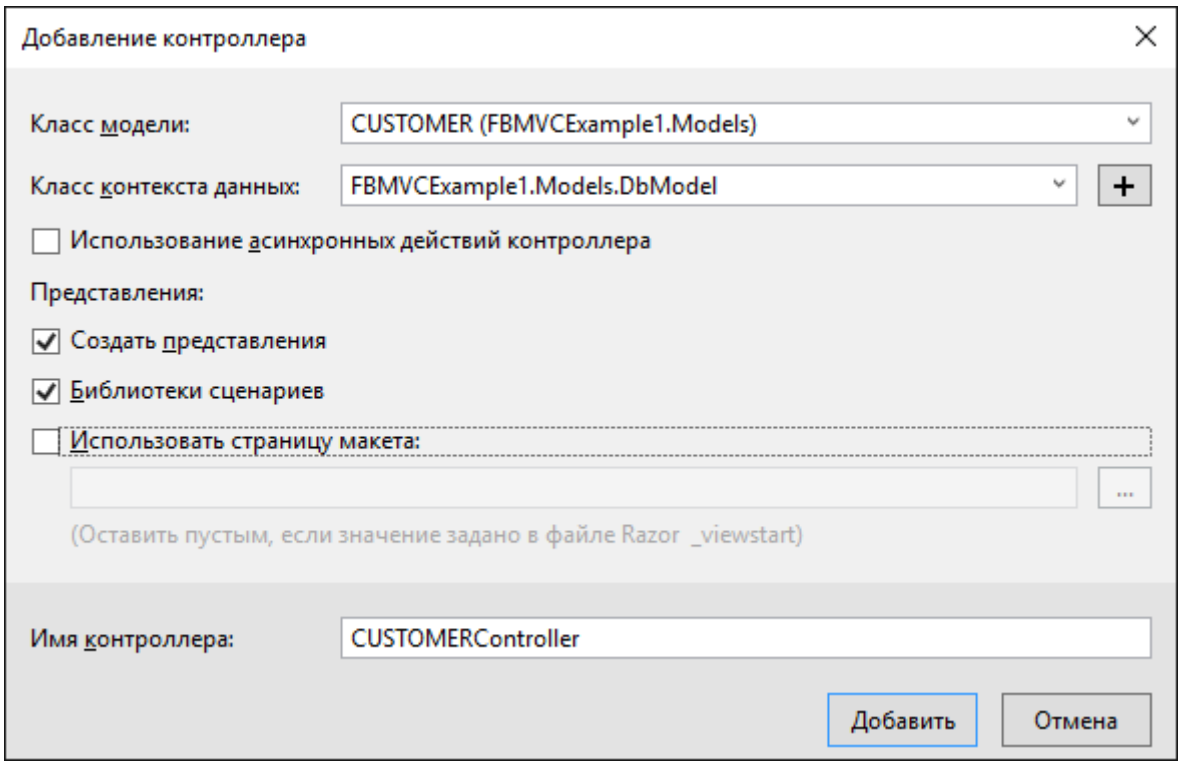
Создание пользовательского интерфейса справочников

Контроллеры

Итак, создадим наш первый контроллер. Он будет служить для отображения и ввода данных о поставщиках.


Просмотреть в браузере (Google Chrome) Ctrl+Shift+W
Просмотреть с помощью...
Контроллер...
Создать элемент... Ctrl+Shift+A
Существующий элемент... Shift+Alt+A
Создать шаблонный элемент...
Создать папку
Добавить папку "ASP.NET"
Azure API App SDK...
Azure API App Client...
New Azure WebJob Project
Existing Project as Azure WebJob
Класс...
Добавить
Открыть элемент как корень обозревателя
Новое представление: Обозреватель решений
Исключить из проекта
Вырезать Ctrl+X
Копировать Ctrl+C
Вставить Ctrl+V
Удалить Del
Переименовать
Открыть папку в проводнике
Свойства Alt+ВВОД


Добавление шаблона
Установлено
Контроллер
Общие
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
Контроллер MVC 5 – пустой
Контроллер MVC 5 с действиями чтения и записи
Контроллер MVC 5 с представлениями, использующий Entity Framework
Контроллер Web API 2 – пустой
Контроллер Web API 2 с действиями чтения и записи
Контроллер Web API 2 с действиями, использующий Entity Framework
Контроллер MVC 5 с представлениями, использующий Entity Framework
по Microsoft
v5.0.0.0
Контроллер MVC с действиями и представлениями Razor для создания, чтения, обновления, удаления и вывода списка сущностей из контекста данных Entity Framework.
Идентификатор: MvcControllerWithContextScaffolder
Щелкните здесь, чтобы выйти в Интернет и найти дополнительные расширения для формирования шаблонов.
Добавить
Отмена



После этих действий будет создан контроллер CustomerController и 5 представлений: для отображения списка поставщиков, детализации поставщика, формы для создания, редактирования и удаления поставщика. Поскольку мы будем активно применять технологию Ajax и библиотеку jqGrid, то нам будет достаточно всего одного представления для отображения списка поставщиков в виде таблицы, остальные действия будут выполняться с помощью jqGrid.

Список поставщиков может оказаться довольно большим. В отличие от настольных приложений в Web приложениях обычно не принято возвращать весь большой список, потому что это может сильно замедлить загрузку страницы. Вместо этого обычно используют постраничное разбиение данных, или динамическую дозагрузку данных, когда при прокрутке пользователь достигает конца страницы (или грида). В нашем примере мы воспользуемся первым вариантом.

Ещё одной особенностью при создании веб приложений является то, что в них отсутствует постоянное соединение с базой данных. Это обусловлено тем, что сам скрипт формирования страницы «живёт» не дольше чем время для формирования ответа на запрос пользователя. Само по себе соединение с базой данных – это довольно дорогой ресурс, поэтому его надо экономить. Конечно, для уменьшения времени установления соединения с базой данных придумали пул соединений, но всё равно желательно, чтобы соединение с базой данных происходило только тогда когда это действительно необходимо. Одним из способов снижения количества взаимодействий с базой данных является перенос проверки правильности введённых данных на сторону браузера. К счастью современные HTML5 и JavaScript библиотеки могут это делать. Например, вы можете проверять обязательность поля на форме ввода, или максимальную длину строковых полей.

Итак давайте изменим контроллер CustomerController для того чтобы он работал с jqGrid. В тексте контроллера сделаны поясняющие комментарии.

```
    public class CustomerController : Controller
    {
        private DbModel db = new DbModel();

        // Отображение представления
        public ActionResult Index()
        {
            return View();
        }

        // Получение данных в виде JSON для грида
        public ActionResult GetData(int? rows, int? page, string sidx, string sord,
            string searchField, string searchString, string searchOper)
        {
            // получаем номер страницы, количество отображаемых данных
            int pageNo = page ?? 1;
            int limit = rows ?? 20;
            // вычисляем смещение
            int offset = (pageNo - 1) * limit;

            // строим запрос для получения поставщиков
            var customersQuery =
                from customer in db.CUSTOMERS
                select new
                {
                    CUSTOMER_ID = customer.CUSTOMER_ID,
                    NAME = customer.NAME,
                    ADDRESS = customer.ADDRESS,
                    ZIPCODE = customer.ZIPCODE,
                    PHONE = customer.PHONE
                };
            // добавлением в запрос условия поиска, если он производится
            if (searchField != null)
            {
                switch (searchOper)
                {
                    case "eq":
                        customersQuery = customersQuery.Where(c => c.NAME ==
searchString);
                        break;
                    case "bw":
                        customersQuery = customersQuery.Where(c =>
c.NAME.StartsWith(searchString));
                        break;
                    case "cn":
                        customersQuery = customersQuery.Where(c =>
c.NAME.Contains(searchString));
                        break;
                }
            }
            // получаем общее количество поставщиков
            int totalRows = customersQuery.Count();

            // добавляем сортировку
            switch (sord) {
                case "asc":
                    customersQuery = customersQuery.OrderBy(customer => customer.NAME);
                    break;
                case "desc":
                    customersQuery = customersQuery.OrderByDescending(customer =>
customer.NAME);
```

```
                    break;
            }

            // получаем список поставщиков
            var customers = customersQuery
                .Skip(offset)
                .Take(limit)
                .ToList();

            // вычисляем общее количество страниц
            int totalPages = totalRows / limit + 1;

            // создаём результат для jqGrid
            var result = new
            {
                page = pageNo,
                total = totalPages,
                records = totalRows,
                rows = customers
            };
            // преобразуем результат в JSON
            return Json(result, JsonRequestBehavior.AllowGet);
        }

        // Добавление нового поставщика
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include = "NAME,ADDRESS,ZIPCODE,PHONE")]
CUSTOMER customer)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                // получаем новый идентификатор с помощью генератора
                customer.CUSTOMER_ID = db.NextValueFor("GEN_CUSTOMER_ID");
                // добавляем модель в список
                db.CUSTOMERS.Add(customer);
                // сохраняем модель
                db.SaveChanges();
                // возвращаем успех в формате JSON
                return Json(true);
            }
            else {
                // соединяем ошибки модели в одну строку
                string messages = string.Join("; ", ModelState.Values
                                        .SelectMany(x => x.Errors)
                                        .Select(x => x.ErrorMessage));
                // возвращаем ошибку в формате JSON
                return Json(new { error = messages });
            }
        }

        // Редактирование поставщика
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include =
"CUSTOMER_ID,NAME,ADDRESS,ZIPCODE,PHONE")] CUSTOMER customer)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                // помечаем модель как изменённую
                db.Entry(customer).State = EntityState.Modified;
                // сохраняем модель
                db.SaveChanges();
```

```
            // возвращаем успех в формате JSON
            return Json(true);
        }
        else {
            // соединяем ошибки модели в одну строку
            string messages = string.Join("; ", ModelState.Values
                                    .SelectMany(x => x.Errors)
                                    .Select(x => x.ErrorMessage));
            // возвращаем ошибку в формате JSON
            return Json(new { error = messages });
        }
    }

    // Удаление поставщика
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id)
    {
        // ищем поставщика по идентификатору
        CUSTOMER customer = db.CUSTOMERS.Find(id);
        // удаляем поставщика
        db.CUSTOMERS.Remove(customer);
        // сохраняем модель
        db.SaveChanges();
        // возвращаем успех в формате JSON
        return Json(true);
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            db.Dispose();
        }
        base.Dispose(disposing);
    }
}
```

Метод Index служит для отображения представления Views/Cusomter/Index.cshtml. Само представление будет представлено чуть позже. В общем, это представление представляет собой шаблон html страницы с разметкой и JavaScript для инициализации jqGrid. Сами данные будут получены в асинхронном режиме в формате JSON с помощью технологии Ajax. В зависимости от выбранной сортировки, номера страницы и параметров поиска формируется HTTP запрос, который будет обработан действием GetData. Параметры http запроса отображаются на входные аргументы метода GetData. В соответствии с этими параметры мы формируем LINQ запрос, и отправляем полученный результат в формате JSON.

Замечание

Для разбора параметров запроса формируемого jqGrid и упрощения построения модели существуют различные библиотеки. Мы не использовали их в наших примерах, и поэтому код может быть несколько громоздким. Однако вы всегда можете улучшить его.

Метод Create предназначен для добавления новой записи о поставщике. Параметры HTTP запроса с типом POST (у метода указан атрибут [HttpPost]) будут отображены на модель Cusotmer. Обратите внимание на строку

```
[Bind(Include = "NAME,ADDRESS,ZIPCODE,PHONE")] CUSTOMER customer
```

Здесь Bind указывает, какие параметры HTTP запроса отображать на свойства модели.

Обратите внимание на атрибут **ValidateAntiforgeryToken**, он предназначен для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия. Наличие этого атрибута требует чтобы в HTTP запросе присутствовал дополнительный параметр __RequestVerificationToken. Этот параметр автоматически добавляется в каждую форму в которой указан хелпер @Html.AntiForgeryToken(). Однако библиотека jqGrid использует динамически формируемые Ajax запросы, а не заранее созданные веб формы. Давайте исправим это. Для этого изменим обобщённое представлении Views/Shared/_Layout.cshtml следующим образом

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title – приложение ASP.NET</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/jquery-ui")

    <link href="~/Content/jquery.jqGrid/ui.jqgrid.css" rel="stylesheet" type="text/css"
/>
    <link href="~/Content/jquery.jqGrid/ui.jqgrid-bootstrap.css" rel="stylesheet"
type="text/css" />
    <link href="~/Content/jquery.jqGrid/ui.jqgrid-bootstrap-ui.css" rel="stylesheet"
type="text/css" />

    <script src="~/Scripts/jquery.jqGrid.min.js" type="text/javascript"></script>
    <script src="~/Scripts/i18n/grid.locale-ru.js" type="text/javascript"></script>
</head>
<body>
    @Html.AntiForgeryToken()
    <script>
        // получение AntiForgery токена
        function GetAntiForgeryToken() {
            var tokenField = $("input[type='hidden'][name$='RequestVerificationToken']");
            if (tokenField.length == 0) {
                return null;
            } else {
                return {
                    name: tokenField[0].name,
                    value: tokenField[0].value
                };
            }
        }

        // добавляем префильтр на все ajax запросы
        // он будет добавлять к любому POST ajax запросу
```

```
        // AntiForgery токен
        $.ajaxPrefilter(
            function (options, localOptions, jqXHR) {
                if (options.type !== "GET") {
                    var token = GetAntiForgeryToken();
                    if (token !== null) {
                        if (options.data.indexOf("X-Requested-With") === -1) {
                            options.data = "X-Requested-With=XMLHttpRequest" +
((options.data === "") ? "" : "&" + options.data);
                        }
                        options.data = options.data + "&" + token.name + '=' +
token.value;
                    }
                }
            }
        );
        // инициализируем общие свойства модуля jqGrid
        $.jgrid.defaults.width = 780;
        $.jgrid.defaults.responsive = true;
        $.jgrid.defaults.styleUI = 'Bootstrap';
    </script>
    <!-- Навигационное меню -->
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Поставщики", "Index", "Customer")</li>
                    <li>@Html.ActionLink("Товары", "Index", "Product")</li>
                    <li>@Html.ActionLink("Накладные", "Index", "Invoice")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year – приложение ASP.NET</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Бандлы

Бандлы предназначены для упрощения подключения JavaScript скриптов и файлов стилей. С помощью хелпера Styles.Render подключаются бандлы стилей, а с помощью хелпера Scripts.Render — бандлы скриптов.

Регистрация бандлов осуществляется в файле *BundleConfig.cs*, который находится в папке *App_Start*:

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                "~/Scripts/jquery.validate*"));

    bundles.Add(new ScriptBundle("~/bundles/jquery-ui").Include(
                "~/Scripts/jquery-ui-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
              "~/Scripts/bootstrap.js",
              "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css").Include(
              "~/Content/jquery-ui.min.css",
              "~/Content/themes/ui-darkness/jquery-ui.min.css",
              "~/Content/themes/ui-darkness/theme.css",
              "~/Content/bootstrap.min.css",
              "~/Content/Site.css"
              ));
}
```

Здесь метод RegisterBundles добавляет все создаваемые бандлы в коллекцию bundles. Объявление бандла выглядит следующим образом: new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-{version}.js").

В конструктор ScriptBundle передаётся виртуальный путь бандла. А с помощью метода Include в данный бандл включаются конкретные файлы скриптов.

В выражении "~/Scripts/jquery-{version}.js" параметр {version} является заменителем для любого символьного обозначения версии скрипта. Это очень удобно, поскольку через некоторое время мы можем поменять версию библиотеки, но при этом в коде нам ничего не придётся менять, так как система уже автоматически примет новую версию.

Выражение "~/Scripts/jquery.validate*" с помощью знака звёздочки заменяет остальную часть строки. Например, это выражение подключит в бандл сразу два файла: jquery.validate.js и jquery.validate.unobtrusive.js (и их минимизированные версии), так как их названия начинаются с jquery.validate*".

То же самое касается и создания бандлов стилей, только в этом случае используется класс StyleBundle.

Во время отладки желательно иметь полные версии скриптов и стилей, а при развёртывании приложения – минифицированные. Бандлы позволяют решить эту задачу. Когда приложение находится в режиме отладки, то файле *web.config* параметр <compilation debug="true">. При изменении этого параметра на значение

false (режим компиляции Release) вместо полных версий JavaScript модулей и файлов CSS стилей будут использоваться минифицированные.

Представления

Из всех автоматически созданных представлений для контроллера Customer нам потребуется только один View/Customer/Index.cshtml, остальные можно удалить из этой папки.

```
@{
    ViewBag.Title = "Index";
}

<h2>Customers</h2>

<table id="jqg"></table>
<div id="jqg-pager"></div>

<script type="text/javascript">
    $(document).ready(function () {

        var dbGrid = $("#jqg").jqGrid({
            url: '@Url.Action("GetData")', // url для получения данных
            datatype: "json", // формат получения данных
            mtype: "GET",     // тип http запроса
            // описание модели
            colModel: [
                {
                    label: 'Id',          // подпись
                    name: 'CUSTOMER_ID',  // имя поля
                    key: true,            // признак ключевого поля
                    hidden: true          // скрыт
                },
                {
                    label: 'Name',
                    name: 'NAME',
                    width: 250,           // ширина
                    sortable: true,       // разрешена сортировка
                    editable: true,       // разрешено редактирование
                    edittype: "text",     // тип поля в редакторе
                    search: true,         // разрешён поиск
                    searchoptions: {
                        sopt: ['eq', 'bw', 'cn'] // разрешённые операторы поиска
                    },
                    editoptions: { size: 30, maxlength: 60 }, // размер и максимальная
длина для поля ввода
                    editrules: { required: true }             // говорит о том что поле
обязательное
                },
                {
                    label: 'Address',
                    name: 'ADDRESS',
                    width: 300,
                    sortable: false,    // запрещаем сортировку
                    editable: true,     // редактируемое
                    search: false,      // запрещаем поиск
                    edittype: "textarea",
                    editoptions: { maxlength: 250, cols: 30, rows: 4 }
                },
                {
                    label: 'Zip Code',
```

```
            name: 'ZIPCODE',
            width: 30,
            sortable: false,
            editable: true,
            search: false,
            edittype: "text",
            editoptions: { size: 30, maxlength: 10 },
        },
        {
            label: 'Phone',
            name: 'PHONE',
            width: 80,
            sortable: false,
            editable: true,
            search: false,
            edittype: "text",
            editoptions: { size: 30, maxlength: 14 },
        }
    ],
    rowNum: 500, // число отображаемых строк
    loadonce: false, // загрузка только один раз
    sortname: 'NAME', // сортировка по умолчанию по столбцу NAME
    sortorder: "asc", // порядок сортировки
    width: window.innerWidth - 80, // ширина грида
    height: 500,      // высота грида
    viewrecords: true, // отображать количество записей
    caption: "Customers", // подпись к гриду
    pager: 'jqg-pager'  // элемент для отображения навигации
});

dbGrid.jqGrid('navGrid', '#jqg-pager', {
    search: true,  // поиск
    add: true,     // добавление
    edit: true,    // редактирование
    del: true,     // удаление
    view: true,    // просмотр записи
    refresh: true, // обновление
    // подписи кнопок
    searchtext: "Поиск",
    addtext: "Добавить",
    edittext: "Изменить",
    deltext: "Удалить",
    viewtext: "Смотреть",
    viewtitle: "Выбранная запись",
    refreshtext: "Обновить"
},
update("edit"), // обновление
update("add"), // добавление
update("del") // удаление
);

// функция возвращающая настройки редактора
function update(act) {
    return {
        closeAfterAdd: true, // закрыть после добавления
        closeAfterEdit: true, // закрыть после редактирования
        width: 400,           // ширина редактора
        reloadAfterSubmit: true, // обновление
        drag: true,              // перетаскиваемый
        // обработчик отправки формы редактирования/удаления/добавления
        onclickSubmit: function (params, postdata) {
            // получаем идентификатор строки
            var selectedRow = dbGrid.getGridParam("selrow");
            // устанавливаем url в зависимости от операции
            switch (act) {
```

```
                        case "add":
                            params.url = '@Url.Action("Create")';
                            break;

                        case "edit":
                            params.url = '@Url.Action("Edit")';
                            postdata.CUSTOMER_ID = selectedRow;
                            break;

                        case "del":
                            params.url = '@Url.Action("Delete")';
                            postdata.CUSTOMER_ID = selectedRow;
                            break;
                    }
                },
                // обработчик результатов обработки форм (операций)
                afterSubmit: function (response, postdata) {
                    var responseData = response.responseJSON;
                    // проверяем результат на наличие сообщений об ошибках
                    if (responseData.hasOwnProperty("error")) {
                        if (responseData.error.length) {
                            return [false, responseData.error];
                        }
                    }
                    else {
                        // обновление грида
                        $(this).jqGrid(
                            'setGridParam',
                            {
                                datatype: 'json'
                            }
                        ).trigger('reloadGrid');
                    }
                    return [true, "", 0];
                }
            };
        };

    });
</script>
```

Как видите всё представление состоит из заголовка, таблицы jqg и блока jqg-pager для отображения панели навигации, остальное занимает скрипт по инициализации грида, панели навигации и диалога редактирования. Для правильного отображения грида, размещения элементов ввода в форме редактирования, настройки валидации форм ввода, настройки возможностей сортировки и поиска важно правильно настроить свойства модели. Эта настройка довольно нетривиальна и содержит множество параметров. Я постарался описать используемые параметры в комментариях. Полное описание параметров модели вы можете найти в документации по библиотеки jqGrid в разделе ColModel API.

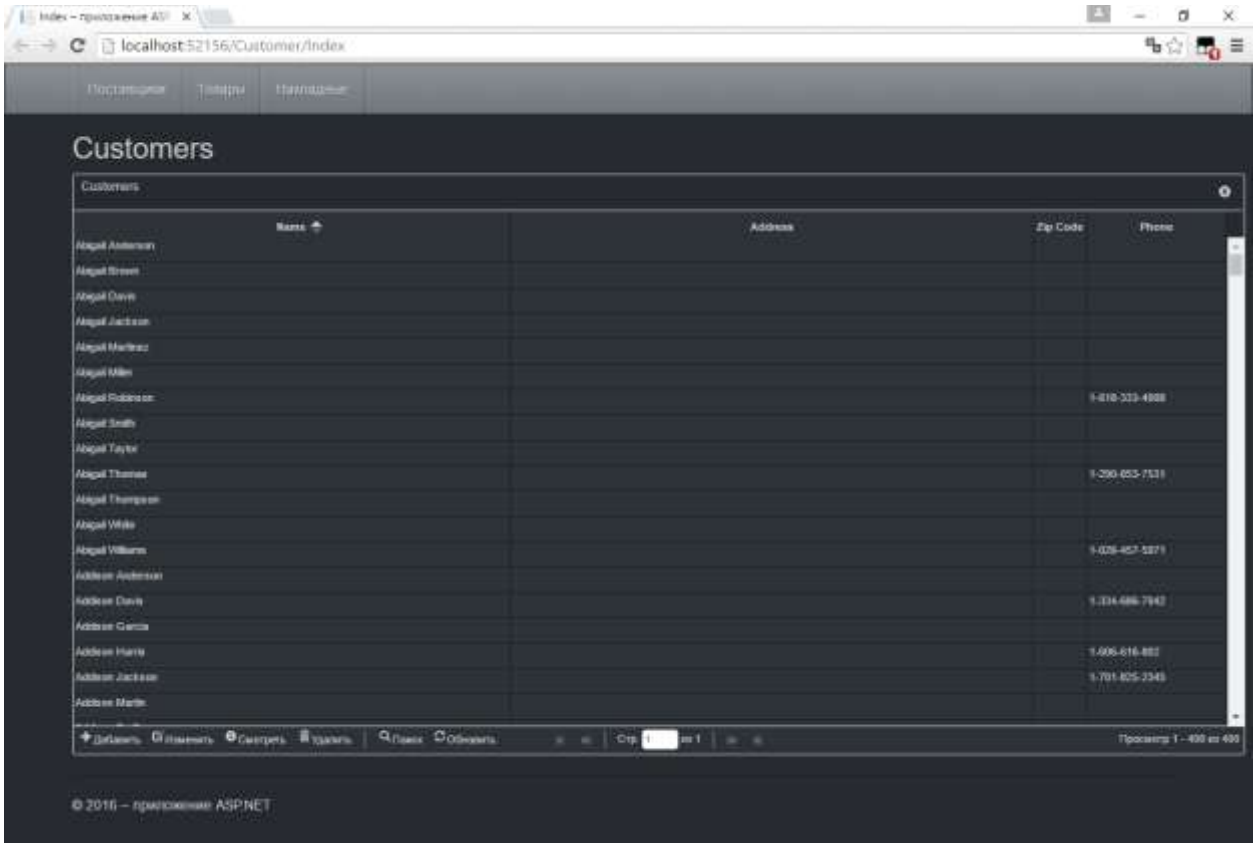
Обратите внимание, что для параметров редактирования и удаления нам пришлось добавить в параметры запроса идентификатор заказчика

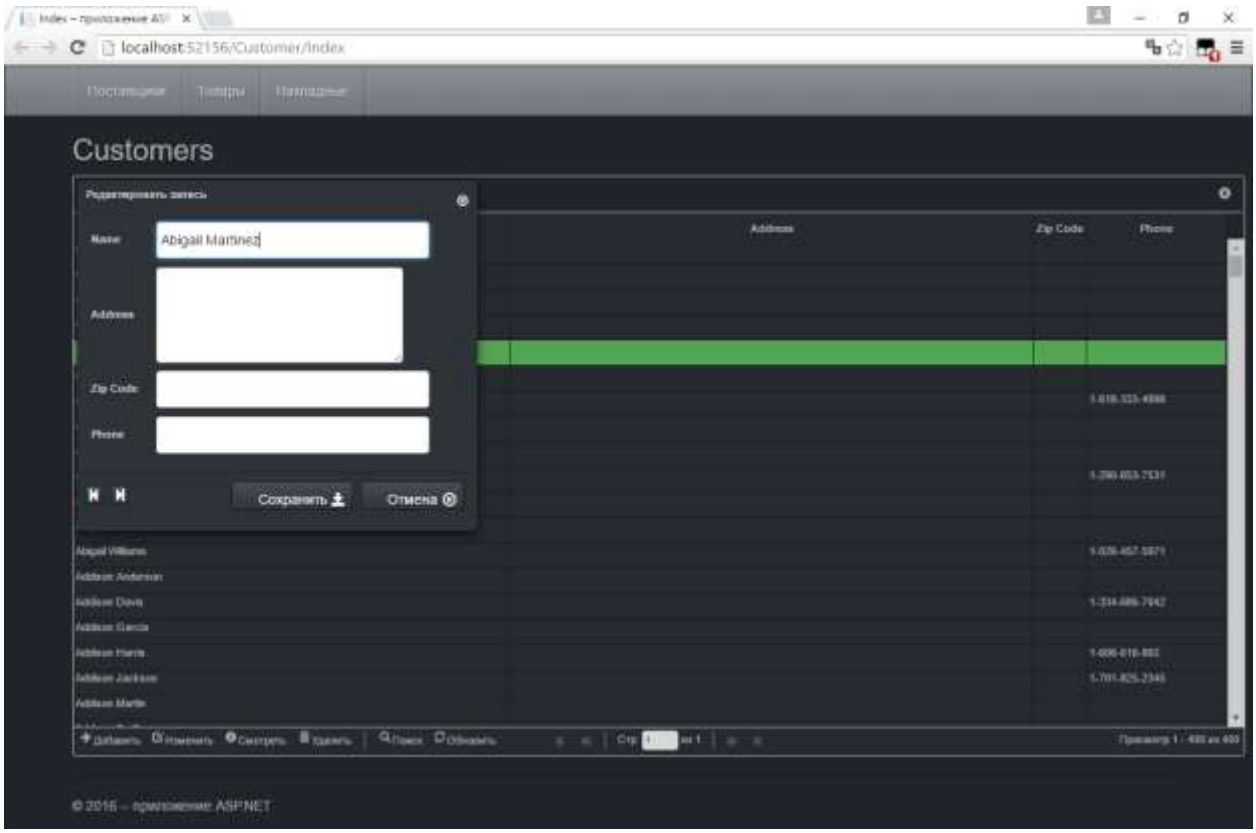
```
                        case "edit":
                            params.url = '@Url.Action("Edit")';
                            postdata.CUSTOMER_ID = selectedRow;
                            break;
```

```
            case "del":
                params.url = '@Url.Action("Delete")';
                postdata.CUSTOMER_ID = selectedRow;
                break;
```

Это сделано потому, что jqGrid автоматически не добавляет в форму ввода скрытые колонки грида, хотя, на мой взгляд, это было бы логично, хотя бы для ключевых полей.

Работающая страница справочника поставщиков будет выглядеть следующим образом:





Контроллер и представление для справочника товаров делаются по аналогии. Здесь мы не будем описывать их подробно, вы можете написать их самостоятельно или найти в исходных кодах, которые прилагаются к данной статье.

Создание пользовательского интерфейса журналов

В нашем приложении будет один журнал «Счёт-фактуры». В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми.

Счёт-фактура – состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и строк счёт-фактуры со списком товаром, их количеством, стоимостью и т.д. Для экономии пространства страницы мы сделаем детализирующий грид скрытым. Он будет отображён лишь при клике по иконке со знаком «+», таким образом, у нас получается, что детализирующий грид вложен в главный.

Контроллеры

Контроллер журнала счёт фактуры должен уметь отдавать данные как по шапкам счёт-фактуры, так и по её позициям. То же самое касается методов для добавления, редактирования и удаления.

```
    public class InvoiceController : Controller
    {
        private DbModel db = new DbModel();

        // Отображение представления
        public ActionResult Index()
        {
            return View();
        }

        // Получение данных в виде JSON для главного грида
        public ActionResult GetData(int? rows, int? page, string sidx, string sord,
            string searchField, string searchString, string searchOper)
        {
            // получаем номер страницы, количество отображаемых данных
            int pageNo = page ?? 1;
            int limit = rows ?? 20;
            // вычисляем смещение
            int offset = (pageNo - 1) * limit;

            // строим запрос для получения счёт-фактур
            var invoicesQuery =
                from invoice in db.INVOICES
                where (invoice.INVOICE_DATE >= AppVariables.StartDate) &&
                      (invoice.INVOICE_DATE <= AppVariables.FinishDate)
                select new
                {
                    INVOICE_ID = invoice.INVOICE_ID,
                    CUSTOMER_ID = invoice.CUSTOMER_ID,
                    CUSTOMER_NAME = invoice.CUSTOMER.NAME,
                    INVOICE_DATE = invoice.INVOICE_DATE,
                    TOTAL_SALE = invoice.TOTAL_SALE,
                    PAID = invoice.PAID
                };

            // добавлением в запрос условия поиска, если он производится
            // для разных полей доступны разные операторы
            // сравнения при поиске
            if (searchField == "CUSTOMER_NAME")
            {
                switch (searchOper)
                {
                    case "eq": // equal
                        invoicesQuery = invoicesQuery.Where(c => c.CUSTOMER_NAME ==
searchString);
                        break;
                    case "bw": // starting with
                        invoicesQuery = invoicesQuery.Where(c =>
c.CUSTOMER_NAME.StartsWith(searchString));
                        break;
                    case "cn": // containing
                        invoicesQuery = invoicesQuery.Where(c =>
c.CUSTOMER_NAME.Contains(searchString));
                        break;
                }
            }
            if (searchField == "INVOICE_DATE")
            {
                var dateValue = DateTime.Parse(searchString);
                switch (searchOper)
```

```
                {
                    case "eq": // =
                        invoicesQuery = invoicesQuery.Where(c => c.INVOICE_DATE ==
dateValue);
                        break;
                    case "lt": // <
                        invoicesQuery = invoicesQuery.Where(c => c.INVOICE_DATE <
dateValue);
                        break;
                    case "le": // <=
                        invoicesQuery = invoicesQuery.Where(c => c.INVOICE_DATE <=
dateValue);
                        break;
                    case "gt": // >
                        invoicesQuery = invoicesQuery.Where(c => c.INVOICE_DATE >
dateValue);
                        break;
                    case "ge": // >=
                        invoicesQuery = invoicesQuery.Where(c => c.INVOICE_DATE >=
dateValue);
                        break;

                }
            }
            if (searchField == "PAID")
            {
                int iVal = (searchString == "on") ? 1 : 0;
                invoicesQuery = invoicesQuery.Where(c => c.PAID == iVal);
            }

            // получаем общее количество счёт-фактур
            int totalRows = invoicesQuery.Count();

            // добавляем сортировку
            switch (sord)
            {
                case "asc":
                    invoicesQuery = invoicesQuery.OrderBy(invoice =>
invoice.INVOICE_DATE);
                    break;
                case "desc":
                    invoicesQuery = invoicesQuery.OrderByDescending(invoice =>
invoice.INVOICE_DATE);
                    break;
            }

            // получаем список счёт-фактур
            var invoices = invoicesQuery
                .Skip(offset)
                .Take(limit)
                .ToList();

            // вычисляем общее количество страниц
            int totalPages = totalRows / limit + 1;

            // создаём результат для jqGrid
            var result = new
            {
                page = pageNo,
                total = totalPages,
                records = totalRows,
                rows = invoices
            };

            // преобразуем результат в JSON
```

```
            return Json(result, JsonRequestBehavior.AllowGet);
        }

        // Получение данных в виде JSON для детализирующего грида
        public ActionResult GetDetailData(int? invoice_id)
        {

            // строим запрос для получения позиций счёт-фактуры
            // отфильтрованный по коду счёт-фактуры
            var lines =
                from line in db.INVOICE_LINES
                where line.INVOICE_ID == invoice_id
                select new
                {
                    INVOICE_LINE_ID = line.INVOICE_LINE_ID,
                    INVOICE_ID = line.INVOICE_ID,
                    PRODUCT_ID = line.PRODUCT_ID,
                    Product = line.PRODUCT.NAME,
                    Quantity = line.QUANTITY,
                    Price = line.SALE_PRICE,
                    Total = Math.Round(line.QUANTITY * line.SALE_PRICE, 2)
                };

            // получаем список позиций
            var invoices = lines
                .ToList();

            // создаём результат для jqGrid
            var result = new
            {
                rows = invoices
            };

            // преобразуем результат в JSON
            return Json(result, JsonRequestBehavior.AllowGet);
        }


        // Добавление новой шапки счёт-фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create([Bind(Include = "CUSTOMER_ID,INVOICE_DATE")] INVOICE
invoice)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                try
                {

                    var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                    var CUSTOMER_ID = new FbParameter("CUSTOMER_ID", FbDbType.Integer);
                    var INVOICE_DATE = new FbParameter("INVOICE_DATE",
FbDbType.TimeStamp);
                    // инициализируем параметры значениями
                    INVOICE_ID.Value = db.NextValueFor("GEN_INVOICE_ID");
                    CUSTOMER_ID.Value = invoice.CUSTOMER_ID;
                    INVOICE_DATE.Value = invoice.INVOICE_DATE;
                    // выполняем ХП
                    db.Database.ExecuteSqlCommand(
                        "EXECUTE PROCEDURE SP_ADD_INVOICE(@INVOICE_ID, @CUSTOMER_ID,
@INVOICE_DATE)",
                        INVOICE_ID,
                        CUSTOMER_ID,
                        INVOICE_DATE);
```

```
                    // возвращаем успех в формате JSON
                    return Json(true);
                }
                catch (Exception ex)
                {
                    // возвращаем ошибку в формате JSON
                    return Json(new { error = ex.Message });
                }

            }
            else {
                string messages = string.Join("; ", ModelState.Values
                                        .SelectMany(x => x.Errors)
                                        .Select(x => x.ErrorMessage));
                // возвращаем ошибку в формате JSON
                return Json(new { error = messages });
            }
        }


        // Редактирование шапки счёт-фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit([Bind(Include = "INVOICE_ID,CUSTOMER_ID,INVOICE_DATE")]
INVOICE invoice)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                try
                {
                    var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                    var CUSTOMER_ID = new FbParameter("CUSTOMER_ID", FbDbType.Integer);
                    var INVOICE_DATE = new FbParameter("INVOICE_DATE",
FbDbType.TimeStamp);
                    // инициализируем параметры значениями
                    INVOICE_ID.Value = invoice.INVOICE_ID;
                    CUSTOMER_ID.Value = invoice.CUSTOMER_ID;
                    INVOICE_DATE.Value = invoice.INVOICE_DATE;
                    // выполняем ХП
                    db.Database.ExecuteSqlCommand(
                        "EXECUTE PROCEDURE SP_EDIT_INVOICE(@INVOICE_ID, @CUSTOMER_ID,
@INVOICE_DATE)",
                        INVOICE_ID,
                        CUSTOMER_ID,
                        INVOICE_DATE);
                    // возвращаем успех в формате JSON
                    return Json(true);
                }
                catch (Exception ex)
                {
                    // возвращаем ошибку в формате JSON
                    return Json(new { error = ex.Message });
                }
            }
            else {
                string messages = string.Join("; ", ModelState.Values
                                        .SelectMany(x => x.Errors)
                                        .Select(x => x.ErrorMessage));
                // возвращаем ошибку в формате JSON
                return Json(new { error = messages });
            }
        }
```

```
        // Удаление шапки счёт-фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Delete(int id)
        {
            try
            {
                var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                // инициализируем параметры значениями
                INVOICE_ID.Value = id;
                // выполняем ХП
                db.Database.ExecuteSqlCommand(
                    "EXECUTE PROCEDURE SP_DELETE_INVOICE(@INVOICE_ID)",
                    INVOICE_ID);
                // возвращаем успех в формате JSON
                return Json(true);
            }
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }

        // Оплата счёт фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Pay(int id)
        {
            try
            {
                var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                // инициализируем параметры значениями
                INVOICE_ID.Value = id;
                // выполняем ХП
                db.Database.ExecuteSqlCommand(
                    "EXECUTE PROCEDURE SP_PAY_FOR_INOVICE(@INVOICE_ID)",
                    INVOICE_ID);
                // возвращаем успех в формате JSON
                return Json(true);
            }
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }

        // Добавление позиции счёт фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult CreateDetail([Bind(Include =
"INVOICE_ID,PRODUCT_ID,QUANTITY")] INVOICE_LINE invoiceLine)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                try
                {
                    var INVOICE_ID = new FbParameter("INVOICE_ID", FbDbType.Integer);
                    var PRODUCT_ID = new FbParameter("PRODUCT_ID", FbDbType.Integer);
                    var QUANTITY = new FbParameter("QUANTITY", FbDbType.Integer);
                    // инициализируем параметры значениями
                    INVOICE_ID.Value = invoiceLine.INVOICE_ID;
                    PRODUCT_ID.Value = invoiceLine.PRODUCT_ID;
```

```
                    QUANTITY.Value = invoiceLine.QUANTITY;
                    // выполняем ХП
                    db.Database.ExecuteSqlCommand(
                        "EXECUTE PROCEDURE SP_ADD_INVOICE_LINE(@INVOICE_ID, @PRODUCT_ID,
@QUANTITY)",
                        INVOICE_ID,
                        PRODUCT_ID,
                        QUANTITY);
                    // возвращаем успех в формате JSON
                    return Json(true);
                }
                catch (Exception ex)
                {
                    // возвращаем ошибку в формате JSON
                    return Json(new { error = ex.Message });
                }

            }
            else {
                string messages = string.Join("; ", ModelState.Values
                                        .SelectMany(x => x.Errors)
                                        .Select(x => x.ErrorMessage));
                // возвращаем ошибку в формате JSON
                return Json(new { error = messages });
            }
        }

        // редактирование позиции счёт фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult EditDetail([Bind(Include =
"INVOICE_LINE_ID,INVOICE_ID,PRODUCT_ID,QUANTITY")] INVOICE_LINE invoiceLine)
        {
            // проверяем правильность модели
            if (ModelState.IsValid)
            {
                try
                {
                    // Создание параметров
                    var INVOICE_LINE_ID = new FbParameter("INVOICE_LINE_ID",
FbDbType.Integer);
                    var QUANTITY = new FbParameter("QUANTITY", FbDbType.Integer);
                    // инициализируем параметры значениями
                    INVOICE_LINE_ID.Value = invoiceLine.INVOICE_LINE_ID;
                    QUANTITY.Value = invoiceLine.QUANTITY;
                    // выполняем ХП
                    db.Database.ExecuteSqlCommand(
                        "EXECUTE PROCEDURE SP_EDIT_INVOICE_LINE(@INVOICE_LINE_ID,
@QUANTITY)",
                        INVOICE_LINE_ID,
                        QUANTITY);
                    // возвращаем успех в формате JSON
                    return Json(true);
                }
                catch (Exception ex)
                {
                    // возвращаем ошибку в формате JSON
                    return Json(new { error = ex.Message });
                }
            }
            else {
                string messages = string.Join("; ", ModelState.Values
                                        .SelectMany(x => x.Errors)
                                        .Select(x => x.ErrorMessage));
                // возвращаем ошибку в формате JSON
```

```
                return Json(new { error = messages });
            }
        }

        // Удаление позиции счёт фактуры
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteDetail(int id)
        {
            try
            {
                // Создание параметров
                var INVOICE_LINE_ID = new FbParameter("INVOICE_LINE_ID",
FbDbType.Integer);
                // инициализируем параметры значениями
                INVOICE_LINE_ID.Value = id;
                // выполняем ХП
                db.Database.ExecuteSqlCommand(
                    "EXECUTE PROCEDURE SP_DELETE_INVOICE_LINE(@INVOICE_LINE_ID)",
                    INVOICE_LINE_ID);
                // возвращаем успех в формате JSON
                return Json(true);
            }
            catch (Exception ex)
            {
                // возвращаем ошибку в формате JSON
                return Json(new { error = ex.Message });
            }
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
```

В методе `GetDetailData` для получения списка позиций счёт-фактуры нет кода для постраничной навигации. Дело в том, что у конкретной счёт-фактуры не очень много позиций для того чтобы применять к ним постраничную навигацию. Это упрощает код, и делает его быстрее.

На этот раз все действия по модификации данных выполняются в хранимых процедурах, однако вы можете выполнить те же действия с помощью Entity Framework. Тексты хранимых процедур вы можете посмотреть в скрипте создания БД.

Представления

Как и для контроллера Customer нам потребуется только одно представление View/Invoice/Index.cshtml, остальные можно удалить из этой папки. Сама разметка представления очень проста, а вот JavaScript кода довольно много. Будем описывать js код по частям.

```
@{
    ViewBag.Title = "Index";
}

<h2>Invoices</h2>

<table id="jqg"></table>
<div id="jpager"></div>

<script type="text/javascript">

    /**
     * Код для работы с jqGrid
     */

</script>
```

Для начала рассмотрим код для работы с главным гридом. По сути, в нём необходимо только прописать свойства модели (типы и размеры полей, параметры поиска, сортировки, видимости и т.д.).

```
        // Грид с инвойсами
        var dbGrid = $("#jqg").jqGrid({
            url: '@Url.Action("GetData")', // url для получения данных
            datatype: "json", // формат получения данных
            mtype: "GET",     // тип http запроса
            // описание модели
            colModel: [
                {
                    label: 'Id',          // подпись
                    name: 'INVOICE_ID',   // имя поля
                    key: true,            // признак ключевого поля
                    hidden: true          // скрыт
                },
                {
                    label: 'CUSTOMER_ID', // подпись
                    name: 'CUSTOMER_ID',  // имя поля
                    hidden: true,         // скрыт
                    editrules: { edithidden: true, required: true }, // скрытое и
требуемое
                    editable: true,       // редактируемое
                    edittype:'custom',    // собственный тип
                    editoptions: {
                        custom_element: function (value, options) {
                            // добавляем скрытый input
                            return $("<input>")
                                .attr('type', 'hidden')
                                .attr('rowid', options.rowId)
                                .addClass("FormElement")
                                .addClass("form-control")
                                .val(value)
                                .get(0);
                        }
                    }
                },
                {
                    label: 'Date',
                    name: 'INVOICE_DATE',
                    width: 60,             // ширина
                    sortable: true,        // позволять сортировку
                    editable: true,        // редактируемое
                    search: true,          // разрешён поиск
                    edittype: "text",      // тип поля ввода
                    align: "right",        // выравнено по правому краю
```

```
        formatter: 'date',     // отформатировано как дата
        sorttype: 'date',      // сортируем как дату
        formatoptions: {       // формат даты
            srcformat: 'd.m.Y H:i:s',
            newformat: 'd.m.Y H:i:s'
        },
        editoptions: {
            // инициализация элемента формы для редактирования
            dataInit: function (element) {
                // создаём datepicker
                $(element).datepicker({
                    id: 'invoiceDate_datePicker',
                    dateFormat: 'dd.mm.yy',
                    minDate: new Date(2000, 0, 1),
                    maxDate: new Date(2030, 0, 1)
                });
            }
        },
        searchoptions: {
            // инициализация элемента формы для поиска
            dataInit: function (element) {
                // создаём datepicker
                $(element).datepicker({
                    id: 'invoiceDate_datePicker',
                    dateFormat: 'dd.mm.yy',
                    minDate: new Date(2000, 0, 1),
                    maxDate: new Date(2030, 0, 1)
                });
            },
            searchoptions: { // типы поиска
                sopt: ['eq', 'lt', 'le', 'gt', 'ge']
            },
        }
    },
    {
        label: 'Customer',
        name: 'CUSTOMER_NAME',
        width: 250,
        editable: true,
        edittype: "text",
        editoptions: {
            size: 50,
            maxlength: 60,
            readonly: true    // только чтение
        },
        editrules: { required: true },
        search: true,
        searchoptions: {
            sopt: ['eq', 'bw', 'cn']
        },
    },
    {
        label: 'Amount',
        name: 'TOTAL_SALE',
        width: 60,
        sortable: false,
        editable: false,
        search: false,
        align: "right",
        formatter: 'currency', // форматировать как валюту
        sorttype: 'number',
        searchrules: {
            "required": true,
            "number": true,
            "minValue": 0
```

```
                    }
                },
                {
                    label: 'Paid',
                    name: 'PAID',
                    width: 30,
                    sortable: false,
                    editable: true,
                    search: true,
                    searchoptions: {
                        sopt: ['eq']
                    },
                    edittype: "checkbox", // галочка
                    formatter: "checkbox",
                    stype: "checkbox",
                    align: "center",
                    editoptions: {
                        value: "1",
                        offval: "0"
                    }
                }
            ],
            rowNum: 500, // число отображаемых строк
            loadonce: false, // загрузка только один раз
            sortname: 'INVOICE_DATE', // сортировка по умолчанию по столбцу NAME
            sortorder: "desc", // порядок сортировки
            width: window.innerWidth - 80, // ширина грида
            height: 500, // высота грида
            viewrecords: true, // отображать количество записей
            caption: "Invoices", // подпись к гриду
            pager: '#jpager', // элемент для отображения постраничной навигации
            subGrid: true, // показывать вложенный грид
            subGridRowExpanded: showChildGrid, // javascript функция для отображения
родительского грида
            subGridOptions: { // опции вложенного грида
                // загружать данные только один раз
                reloadOnExpand: false,
                // загружать строки подгрида только при щелчке по иконке "+"
                selectOnExpand: true
            },
        });

        // отображение панели навигации
        dbGrid.jqGrid('navGrid', '#jpager',
            {
                search: true,  // поиск
                add: true,     // добавление
                edit: true,    // редактирование
                del: true,     // удаление
                view: false,    // просмотр записи
                refresh: true, // обновление

                searchtext: "Поиск",
                addtext: "Добавить",
                edittext: "Изменить",
                deltext: "Удалить",
                viewtext: "Смотреть",
                viewtitle: "Выбранная запись",
                refreshtext: "Обновить"
           },
           update("edit"), // обновление
           update("add"), // добавление
           update("del") // удаление
        );
```

Добавим в главный грид ещё «пользовательскую» одну кнопку для оплаты счёт-фактуры.

```
        // добавление кнопки для оплаты счёт фактуры
        dbGrid.navButtonAdd('#jpager',
            {
                buttonicon: "glyphicon-usd",
                title: "Оплатить",
                caption: "Оплатить",
                position: "last",
                onClickButton: function () {
                    // получаем идентификатор текущей записи
                    var id = dbGrid.getGridParam("selrow");
                    if (id) {
                        var url = '@Url.Action("Pay")';
                        $.ajax({
                            url: url,
                            type: 'POST',
                            data: { id: id },
                            success: function (data) {
                                // проверяем, не произошла ли ошибка
                                if (data.hasOwnProperty("error")) {
                                    alertDialog('Ошибка', data.error);
                                }
                                else {
                                    // обновление грида
                                    $("#jqg").jqGrid(
                                        'setGridParam',
                                        {
                                            datatype: 'json'
                                        }
                                    ).trigger('reloadGrid');
                                }
                            }
                        });
                    }
                }
            });
```

В отличие от справочников диалоги редактирования для журналов намного сложнее. Зачастую они используют выбор из других справочников. Поэтому такие диалоги редактирования не получится построить стандартными способами jqGrid, однако в этой библиотеки существует возможность построение диалогов по шаблону, которой мы и воспользуемся.

Для выбора заказчика сделаем поле только для чтения и разместим справа от него кнопку для вызова формы с гридом для отображения списка заказчиков.

```
        // возвращает свойства для создания диалогов редактирования
        function update(act) {
            // шаблон диалога редактирования
            var template = "<div style='margin-left:15px;' id='dlgEditInvoice'>";
            template += "<div>{CUSTOMER_ID} </div>";
            template += "<div> Date: </div><div>{INVOICE_DATE} </div>";
            // поле ввода заказчика с кнопкой
            template += "<div> Customer <sup>*</sup>:</div>";
            template += "<div>";
            template += "<div style='float: left;'>{CUSTOMER_NAME}</div> ";
            template += "<a style='margin-left: 0.2em;' class='btn'
onclick='showCustomerWindow(); return false;'>";
```

```
            template += "<span class='glyphicon glyphicon-folder-open'></span>
Выбрать</a> ";
            template += "<div style='clear: both;'></div>";
            template += "</div>";
            template += "<div> {PAID} Paid </div>";
            template += "<hr style='width: 100%;'/>";
            template += "<div> {sData} {cData}  </div>";
            template += "</div>";

            return {
                top: $(".container.body-content").position().top + 150,
                left: $(".container.body-content").position().left + 150,
                modal: true,
                drag: true,
                closeOnEscape: true,
                closeAfterAdd: true, // закрыть после добавления
                closeAfterEdit: true, // закрыть после редактирования
                reloadAfterSubmit: true, // обновление
                template: (act != "del") ? template : null,
                onclickSubmit: function (params, postdata) {
                    // получаем идентификатор строки
                    var selectedRow = dbGrid.getGridParam("selrow");
                    switch (act) {
                        case "add":
                            params.url = '@Url.Action("Create")';
                            // получаем идентификатор заказчика для текущей строки
                            postdata.CUSTOMER_ID = $('#dlgEditInvoice
input[name=CUSTOMER_ID]').val();
                            break;

                        case "edit":
                            params.url = '@Url.Action("Edit")';
                            postdata.INVOICE_ID = selectedRow;
                            // получаем идентификатор заказчика для текущей строки
                            postdata.CUSTOMER_ID = $('#dlgEditInvoice
input[name=CUSTOMER_ID]').val();
                            break;

                        case "del":
                            params.url = '@Url.Action("Delete")';
                            postdata.INVOICE_ID = selectedRow;
                            break;
                    }
                },
                afterSubmit: function (response, postdata) {
                    var responseData = response.responseJSON;
                    // проверяем результат на наличие сообщений об ошибках
                    if (responseData.hasOwnProperty("error")) {
                        if (responseData.error.length) {
                            return [false, responseData.error];
                        }
                    }
                    else {
                        // обновление грида
                        $(this).jqGrid(
                            'setGridParam',
                            {
                                datatype: 'json'
                            }
                        ).trigger('reloadGrid');
                    }
                    return [true, "", 0];
                }
            };
        };
```

Теперь напишем функцию для открытия справочника заказчиков. В этой функции мы будем создавать диалог с помощью библиотеки Bootstrap, в котором будет размещён грид для выбора заказчика. По сути, это тот же самый грид, который мы использовали выше, но размещённый внутри диалогового окна. При нажатии кнопки «OK» идентификатор заказчика и его имя будут записаны в элементы ввода родительского диалога для редактирования счёт-фактуры.

```
/**
 * Отображение окна для выбора справочника заказчиков
 */
function showCustomerWindow() {
    // основной блок диалога
    var dlg = $('<div>')
        .attr('id', 'dlgChooseCustomer')
        .attr('aria-hidden', 'true')
        .attr('role', 'dialog')
        .attr('data-backdrop', 'static')
        .css("z-index", '2000')
        .addClass('modal')
        .appendTo($('body'));

    // блок с содержимым диалога
    var dlgContent = $("<div>")
        .addClass("modal-content")
        .css('width', '730px')
        .appendTo($('<div>')
                        .addClass('modal-dialog')
                        .appendTo(dlg));

    // блок с шапкой диалога
    var dlgHeader = $('<div>').addClass("modal-header").appendTo(dlgContent);
    // кнопка "X" для закрытия
    $("<button>")
        .addClass("close")
        .attr('type', 'button')
        .attr('aria-hidden', 'true')
        .attr('data-dismiss', 'modal')
        .html("&times;")
        .appendTo(dlgHeader);
    // подпись
    $("<h5>").addClass("modal-title").html("Выбор заказчика").appendTo(dlgHeader);

    // тело диалога
    var dlgBody = $('<div>')
        .addClass("modal-body")
        .appendTo(dlgContent);

    // подвал диалога
    var dlgFooter = $('<div>').addClass("modal-footer").appendTo(dlgContent);
    // Кнопка "OK"
    $("<button>")
        .attr('type', 'button')
        .addClass('btn')
        .html('OK')
        .on('click', function () {
            var rowId = $("#jqgCustomer").jqGrid("getGridParam", "selrow");
            var row = $("#jqgCustomer").jqGrid("getRowData", rowId);
            // сохраняем идентификатор и имя заказчика
            // в элементы ввода родительской формы
            $('#dlgEditInvoice input[name=CUSTOMER_ID]').val(rowId);
            $('#dlgEditInvoice input[name=CUSTOMER_NAME]').val(row["NAME"]);
            dlg.modal('hide');
        })
```

```
            .appendTo(dlgFooter);
        // Кнопка "Cancel"
        $("<button>")
            .attr('type', 'button')
            .addClass('btn')
            .html('Cancel')
            .on('click', function () { dlg.modal('hide'); })
            .appendTo(dlgFooter);
        // добавляем таблицу для отображения заказчиков в тело диалога
        $('<table>')
            .attr('id', 'jqgCustomer')
            .appendTo(dlgBody);
        // добавляем панель навигации
        $('<div>')
            .attr('id', 'jqgCustomerPager')
            .appendTo(dlgBody);

        dlg.on('hidden.bs.modal', function () {
            dlg.remove();
        });

        // отображаем диалог
        dlg.modal();

        // создание и инициализация jqGrid
        var dbGrid = $("#jqgCustomer").jqGrid({
            url: '@Url.Action("GetData", "Customer")', // url для получения данных
            mtype: "GET", // тип http запроса
            datatype: "json", // формат получения данных
            page: 1,
            width: '100%',
            // описание модели
            colModel: [
                {
                    label: 'Id',          // подпись
                    name: 'CUSTOMER_ID',  // имя поля
                    key: true,            // признак ключевого поля
                    hidden: true          // скрытое
                },
                {
                    label: 'Name',
                    name: 'NAME',
                    width: 250,           // ширина
                    sortable: true,       // разрешена сортировка
                    editable: true,       // разрешено редактирование
                    edittype: "text",     // тип поля в редакторе
                    search: true,         // разрешён поиск
                    searchoptions: {
                        sopt: ['eq', 'bw', 'cn'] // разрешённые операторы поиска
                    },
                    editoptions: { size: 30, maxlength: 60 }, // размер и максимальная
длина для поля ввода
                    editrules: { required: true }    // говорит о том что поле
обязательное
                },
                {
                    label: 'Address',
                    name: 'ADDRESS',
                    width: 300,
                    sortable: false,      // запрещаем сортировку
                    editable: true,       // редактируемое
                    search: false,        // запрещаем поиск
                    edittype: "textarea",
                    editoptions: { maxlength: 250, cols: 30, rows: 4 }
                },
```

```
            {
                label: 'Zip Code',
                name: 'ZIPCODE',
                width: 60,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: { size: 30, maxlength: 10 },
            },
            {
                label: 'Phone',
                name: 'PHONE',
                width: 85,
                sortable: false,
                editable: true,
                search: false,
                edittype: "text",
                editoptions: { size: 30, maxlength: 14 },
            }
        ],
        loadonce: false,
        pager: '#jqgCustomerPager',
        rowNum: 500, // число отображаемых строк
        sortname: 'NAME', // сортировка по умолчанию по столбцу NAME
        sortorder: "asc", // порядок сортировки
        height: 500
    });

    dbGrid.jqGrid('navGrid', '#jqgCustomerPager',
        {
            search: true,  // поиск
            add: false,     // добавление
            edit: false,    // редактирование
            del: false,     // удаление
            view: false,    // просмотр записи
            refresh: true, // обновление

            searchtext: "Поиск",
            viewtext: "Смотреть",
            viewtitle: "Выбранная запись",
            refreshtext: "Обновить"
        }
    );
}
```

Для этого журнала нам осталось написать функцию `showChildGrid`, которая позволяет просматривать и редактировать информацию о позициях накладной. Эта функция будет динамически создавать грид с позициями счёт-фактуры при нажатии на кнопку «+» (для раскрытия деталей). Для загрузки данных о позиции нам будет необходимо передавать первичный ключ выбранной шапки счёт фактуры.

```
// обработчик события раскрытия родительского грида
// принимает два параметра идентификатор родительской записи
// и первичный ключ записи
function showChildGrid(parentRowID, parentRowKey) {
    var childGridID = parentRowID + "_table";
    var childGridPagerID = parentRowID + "_pager";

    // отправляем первичный ключ родительской записи
    // чтобы отфильтровать записи позиций накладной
```

```
            var childGridURL = '@Url.Action("GetDetailData")';
            childGridURL = childGridURL + "?invoice_id=" +
encodeURIComponent(parentRowKey)

            // добавляем HTML элементы для отображения таблицы и постраничной навигации
            // как дочерние для выбранной строки в мастер гриде
            $('<table>')
                .attr('id', childGridID)
                .appendTo($('#' + parentRowID));
            $('<div>')
                .attr('id', childGridPagerID)
                .addClass('scroll')
                .appendTo($('#' + parentRowID));

            // создаём и инициализируем дочерний грид
            var detailGrid = $("#" + childGridID).jqGrid({
                url: childGridURL,
                mtype: "GET",
                datatype: "json",
                page: 1,
                colModel: [
                    {
                        label: 'Invoice Line ID',
                        name: 'INVOICE_LINE_ID',
                        key: true,
                        hidden: true
                    },
                    {
                        label: 'Invoice ID',
                        name: 'INVOICE_ID',
                        hidden: true,
                        editrules: { edithidden: true, required: true },
                        editable: true,
                        edittype: 'custom',
                        editoptions: {
                            custom_element: function (value, options) {
                                // создаём скрытый элемент ввода
                                return $("<input>")
                                    .attr('type', 'hidden')
                                    .attr('rowid', options.rowId)
                                    .addClass("FormElement")
                                    .addClass("form-control")
                                    .val(parentRowKey)
                                    .get(0);
                            }
                        }
                    },
                    {
                        label: 'Product ID',
                        name: 'PRODUCT_ID',
                        hidden: true,
                        editrules: { edithidden: true, required: true },
                        editable: true,
                        edittype: 'custom',
                        editoptions: {
                            custom_element: function (value, options) {
                                // создаём скрытый элемент ввода
                                return $("<input>")
                                    .attr('type', 'hidden')
                                    .attr('rowid', options.rowId)
                                    .addClass("FormElement")
                                    .addClass("form-control")
                                    .val(value)
                                    .get(0);
                            }
```

```
                        }
                    },
                    {
                        label: 'Product',
                        name: 'Product',
                        width: 300,
                        editable: true,
                        edittype: "text",
                        editoptions: {
                            size: 50,
                            maxlength: 60,
                            readonly: true
                        },
                        editrules: { required: true }
                    },
                    {
                        label: 'Price',
                        name: 'Price',
                        formatter: 'currency',
                        editable: true,
                        editoptions: {
                            readonly: true
                        },
                        align: "right",
                        width: 100
                    },
                    {
                        label: 'Quantity',
                        name: 'Quantity',
                        align: "right",
                        width: 100,
                        editable: true,
                        editrules: { required: true, number: true, minValue: 1 },
                        editoptions: {
                            dataEvents: [
                                {
                                    type: 'change',
                                    fn: function (e) {
                                        var quantity = $(this).val() - 0;
                                        var price = $('#dlgEditInvoiceLine
input[name=Price]').val() - 0;
                                        $('#dlgEditInvoiceLine
input[name=Total]').val(quantity * price)
                                    }
                                }
                            ],
                            defaultValue: 1
                        }
                    },
                    {
                        label: 'Total',
                        name: 'Total',
                        formatter: 'currency',
                        align: "right",
                        width: 100,
                        editable: true,
                        editoptions: {
                            readonly: true
                        }
                    }
                ],
                loadonce: false,
                width: '100%',
                height: '100%',
                pager: "#" + childGridPagerID
```

```
            });

            // отображение панели инструментов
            $("#" + childGridID).jqGrid('navGrid', '#' + childGridPagerID,
                {
                    search: false,  // поиск
                    add: true,     // добавление
                    edit: true,    // редактирование
                    del: true,     // удаление
                    refresh: true // обновление
                },
                updateDetail("edit"), // обновление
                updateDetail("add"), // добавление
                updateDetail("del") // удаление
            );

            // функция возвращающая настройки для диалога редактирования
            function updateDetail(act) {
                // шаблон диалога редактирования
                var template = "<div style='margin-left:15px;' id='dlgEditInvoiceLine'>";
                template += "<div>{INVOICE_ID} </div>";
                template += "<div>{PRODUCT_ID} </div>";
                // поле ввода товара с кнопкой
                template += "<div> Product <sup>*</sup>:</div>";
                template += "<div>";
                template += "<div style='float: left;'>{Product}</div> ";
                template += "<a style='margin-left: 0.2em;' class='btn'
onclick='showProductWindow(); return false;'>";
                template += "<span class='glyphicon glyphicon-folder-open'></span>
Выбрать</a> ";
                template += "<div style='clear: both;'></div>";
                template += "</div>";
                template += "<div> Quantity: </div><div>{Quantity} </div>";
                template += "<div> Price: </div><div>{Price} </div>";
                template += "<div> Total: </div><div>{Total} </div>";
                template += "<hr style='width: 100%;'/>";
                template += "<div> {sData} {cData}  </div>";
                template += "</div>";

                return {
                    top: $(".container.body-content").position().top + 150,
                    left: $(".container.body-content").position().left + 150,
                    modal: true,
                    drag: true,
                    closeOnEscape: true,
                    closeAfterAdd: true, // закрыть после добавления
                    closeAfterEdit: true, // закрыть после редактирования
                    reloadAfterSubmit: true, // обновление
                    template: (act != "del") ? template : null,
                    onclickSubmit: function (params, postdata) {
                        var selectedRow = detailGrid.getGridParam("selrow");
                        switch (act) {
                            case "add":
                                params.url = '@Url.Action("CreateDetail")';
                                // получаем идентификатор счёт-фактуры
                                postdata.INVOICE_ID = $('#dlgEditInvoiceLine
input[name=INVOICE_ID]').val();
                                // получаем идентификатор товара для текущей записи
                                postdata.PRODUCT_ID = $('#dlgEditInvoiceLine
input[name=PRODUCT_ID]').val();
                                break;

                            case "edit":
                                params.url = '@Url.Action("EditDetail")';
                                // получаем идентификатор текущей записи
```

```
                        postdata.INVOICE_LINE_ID = selectedRow;
                        break;

                    case "del":
                        params.url = '@Url.Action("DeleteDetail")';
                        // получаем идентификатор текущей записи
                        postdata.INVOICE_LINE_ID = selectedRow;
                        break;
                }
            },
            afterSubmit: function (response, postdata) {
                var responseData = response.responseJSON;
                // проверяем результат на наличие сообщений об ошибках
                if (responseData.hasOwnProperty("error")) {
                    if (responseData.error.length) {
                        return [false, responseData.error];
                    }
                }
                else {
                    // обновление грида
                    $(this).jqGrid(
                        'setGridParam',
                        {
                            datatype: 'json'
                        }
                    ).trigger('reloadGrid');
                }
                return [true, "", 0];
            }
        };
    };
```

Вот теперь создание журнала счёт-фактур закончено. Здесь мы не рассмотрели функцию `showProductWindow`, которая предназначена для выбора товара из справочника при заполнении позиций счёт-фактуры. Эта функция полностью аналогична ранее описанной функции `'showCustomerWindow`, предназначенной для выбора из справочника заказчиков.

Внимательный читатель мог заметить, что функции для отображения выбора из справочника и отображения справочника почти идентичные. Это можно улучшить выносив эти функции в отдельные файлы скриптов с расширением js. Попробуйте сделать это самостоятельно.

Аутентификация и авторизация

Технология ASP.NET имеет мощный механизм для организации авторизации и аутентификации в .NET приложениях под названием ASP.NET Identity. Инфраструктура OWIN и AspNet Identity позволяют производить как стандартную авторизацию, так и авторизацию через внешние сервисы с помощью аккаунтов в Google, Twitter, Facebook и т.д. Описание технологии ASP.NET Identity является достаточно объёмным и выходит за рамки данной статьи. Вы можете почитать об этой технологии на сайте http://www.asp.net/identity.

А в нашем приложении мы будем использовать чуть более простую модель, основанную на аутентификации форм. Для включения аутентификации форм необходимо сделать изменения в файле конфигурации *web.config*. Находим секцию <system.web> и внутри этой секции поместим следующую подсекцию:

```
<authentication mode="Forms">
      <forms name="cookies" timeout="2880" loginUrl="~/Account/Login"
defaultUrl="~/Invoice/Index"/>
</authentication>
```

Установив mode="Forms", мы тем самым подключаем аутентификацию форм. Далее мы задаём ряд параметров. Нам доступен следующий список параметров:

- **coockieless**: определяет, применяются ли куки-наборы и как они используются. Может принимать следующие значения: **UseCookies**(определяет, что куки-наборы будут использоваться всегда вне зависимости от устройства), **UseUri** (куки-наборы никогда не используются), **AutoDetect** (если устройство поддерживает куки-наборы, то они используются, в противном случае они не применяются, при этом проводится тестирование, определяющее, включена ли поддержка), **UseDeviceProfile** (если устройство поддерживает куки-наборы, то они используются, в противном случае они не применяются, в отличие от предыдущего случая тестирование не проводится. Используется по умолчанию).
- **defaultUrl**: определяет путь, по которому осуществляется переход после авторизации
- **domain**: определяет куки-наборы для всего домена. Благодаря этому мы можем использовать одни и те же куки-наборы для главного домена и его субдоменов. По умолчанию имеет значение в качестве пустой строки
- **loginUrl**: адрес для аутентификации пользователя. Значение по умолчанию - "~/Account/Login"
- **name**: задаёт имя для куки-набора. Значение по умолчанию - ".ASPXAUTH"
- **path**: задаёт путь для куки-наборов. Значение по умолчанию - "/"
- **requireSSL**: определяет, требуется ли SSL-соединение для передачи куки-наборов. Значение по умолчанию false
- **timeout**: определяет срок действия куков в минутах

В нашем приложении мы будем хранить данные аутентификации в той же базе данных, что и другие данные, поэтому настройка дополнительной строки подключения нам не потребуется.

Теперь надо создать всю необходимую инфраструктуру для аутентификации - модели, контроллеры и представления. Создадим модель WebUser, которая будет описывать пользователя:

```
    [Table("Firebird.WEBUSER")]
    public partial class WEBUSER
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
```

```
        public WEBUSER()
        {
            WEBUSERINROLES = new HashSet<WEBUSERINROLE>();
        }

        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int WEBUSER_ID { get; set; }

        [Required]
        [StringLength(63)]
        public string EMAIL { get; set; }

        [Required]
        [StringLength(63)]
        public string PASSWD { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<WEBUSERINROLE> WEBUSERINROLES { get; set; }
    }
```

Добавим ещё две модели: одну для описания ролей WEBROLE, и одну для связи ролей с пользователями WEBUSERINROLE.

```
    [Table("Firebird.WEBROLE")]
    public partial class WEBROLE
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int WEBROLE_ID { get; set; }

        [Required]
        [StringLength(63)]
        public string NAME { get; set; }

    }

    [Table("Firebird.WEBUSERINROLE")]
    public partial class WEBUSERINROLE
    {

        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int ID { get; set; }

        [Required]
        public int WEBUSER_ID { get; set; }

        [Required]
        public int WEBROLE_ID { get; set; }

        public virtual WEBUSER WEBUSER { get; set; }

        public virtual WEBROLE WEBROLE { get; set; }
    }
```

В классе DbModel с помощью Fluent API укажем связи между WEBUSER и WEBUSERINROLE.

…

```
        public virtual DbSet<WEBUSER> WEBUSERS { get; set; }
        public virtual DbSet<WEBROLE> WEBROLES { get; set; }
        public virtual DbSet<WEBUSERINROLE> WEBUSERINROLES { get; set; }
…
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<WEBUSER>()
                .HasMany(e => e.WEBUSERINROLES)
                .WithRequired(e => e.WEBUSER)
                .WillCascadeOnDelete(false);

…
        }

…
```

Поскольку мы используем технологию Database First, то таблицы в БД могут быть созданы автоматически, но я предпочитаю сам контролировать этот процесс, поэтому приведу здесь скрипт создания дополнительных таблиц.

```
RECREATE TABLE WEBUSER (
  WEBUSER_ID INT NOT NULL,
  EMAIL VARCHAR(63) NOT NULL,
  PASSWD VARCHAR(63) NOT NULL,
  CONSTRAINT PK_WEBUSER PRIMARY KEY(WEBUSER_ID),
  CONSTRAINT UNQ_WEBUSER UNIQUE(EMAIL)
);

RECREATE TABLE WEBROLE (
  WEBROLE_ID INT NOT NULL,
  NAME VARCHAR(63) NOT NULL,
  CONSTRAINT PK_WEBROLE PRIMARY KEY(WEBROLE_ID),
  CONSTRAINT UNQ_WEBROLE UNIQUE(NAME)
);

RECREATE TABLE WEBUSERINROLE (
  ID INT NOT NULL,
  WEBUSER_ID INT NOT NULL,
  WEBROLE_ID INT NOT NULL,
  CONSTRAINT PK_WEBUSERINROLE PRIMARY KEY(ID)
);


ALTER TABLE WEBUSERINROLE
ADD CONSTRAINT FK_WEBUSERINROLE_USER FOREIGN KEY (WEBUSER_ID) REFERENCES
WEBUSER (WEBUSER_ID);

ALTER TABLE WEBUSERINROLE
ADD CONSTRAINT FK_WEBUSERINROLE_ROLE FOREIGN KEY (WEBROLE_ID) REFERENCES
WEBROLE (WEBROLE_ID);


RECREATE SEQUENCE SEQ_WEBUSER;
RECREATE SEQUENCE SEQ_WEBROLE;
RECREATE SEQUENCE SEQ_WEBUSERINROLE;

SET TERM ^;

RECREATE TRIGGER TBI_WEBUSER
FOR WEBUSER
ACTIVE BEFORE INSERT
AS
BEGIN
```

```
  IF (NEW.WEBUSER_ID IS NULL) THEN
    NEW.WEBUSER_ID = NEXT VALUE FOR SEQ_WEBUSER;
END^

RECREATE TRIGGER TBI_WEBROLE
FOR WEBROLE
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW.WEBROLE_ID IS NULL) THEN
    NEW.WEBROLE_ID = NEXT VALUE FOR SEQ_WEBROLE;
END^

RECREATE TRIGGER TBI_WEBUSERINROLE
FOR WEBUSERINROLE
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = NEXT VALUE FOR SEQ_WEBUSERINROLE;
END^

SET TERM ;^
```

Добавим два пользователя и две роли для проверки.

```
INSERT INTO WEBUSER (EMAIL, PASSWD) VALUES ('john', '12345');
INSERT INTO WEBUSER (EMAIL, PASSWD) VALUES ('alex', '123');

COMMIT;

INSERT INTO WEBROLE (NAME) VALUES ('admin');
INSERT INTO WEBROLE (NAME) VALUES ('manager');

COMMIT;

-- Связываем пользователей и роли
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(1, 1);
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(1, 2);
INSERT INTO WEBUSERINROLE(WEBUSER_ID, WEBROLE_ID) VALUES(2, 2);

COMMIT;
```

> **Замечание о паролях**
>
> Обычно вместо пароля в открытом виде хранят некий хэш от него, например, по алгоритму md5. В нашем примере мы немного упростили аутентификацию.

При регистрации и логине мы не будем напрямую взаимодействовать с моделью WebUser. Вместо этого мы будем использовать специальные модели, которые также добавим в проект:

```
namespace FBMVCExample.Models
{
    using System;
    using System.Collections.Generic;
```

```
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using System.Data.Entity.Spatial;

    // Модель для входа в систему
    public class LoginModel
    {
        [Required]
        public string Name { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }

    // Модель для регистрации нового пользователя
    public class RegisterModel
    {
        [Required]
        public string Name { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Compare("Password", ErrorMessage = "Пароли не совпадают")]
        public string ConfirmPassword { get; set; }

    }

}
```

Эти модели будет использоваться соответственно для представлений логина и регистрации. Эти представление для входа будет выглядеть следующим образом:

```
@model FBMVCExample.Models.LoginModel

@{
    ViewBag.Title = "Вход";
}

<h2>Вход</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        @Html.ValidationSummary(true)

        <div class="form-group">
            @Html.LabelFor(model => model.Name, new { @class = "control-label col-md-2"
})
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
```

```
            @Html.LabelFor(model => model.Password, new { @class = "control-label col-md-
2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Password)
                @Html.ValidationMessageFor(model => model.Password)
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Вход" class="btn btn-default" />
            </div>
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Соответственно, представление для регистрации будет выглядеть так:

```
@model FBMVCExample.Models.RegisterModel

@{
    ViewBag.Title = "Регистрация";
}

<h2>Регистрация</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        @Html.ValidationSummary(true)

        <div class="form-group">
            @Html.LabelFor(model => model.Name, new { @class = "control-label col-md-2"
})
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Password, new { @class = "control-label col-md-
2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Password)
                @Html.ValidationMessageFor(model => model.Password)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.ConfirmPassword, new { @class = "control-label
col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ConfirmPassword)
                @Html.ValidationMessageFor(model => model.ConfirmPassword)
            </div>
        </div>
```

```
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Зарегистрировать" class="btn btn-default" />
            </div>
        </div>
    </div>
}

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Замечание о пользователях

В данном примере модель, представление и контроллеры для входа и регистрации пользователей предельно упрощены, т.к. обычно пользователь имеет существенно больше атрибутов, чем логин и пароль.

Теперь добавим новый контроллер AccountController со следующим содержанием:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Security;
using FBMVCExample.Models;

namespace FBMVCExample.Controllers
{
    public class AccountController : Controller
    {
        public ActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Login(LoginModel model)
        {
            if (ModelState.IsValid)
            {
                // поиск пользователя в бд
                WEBUSER user = null;
                using (DbModel db = new DbModel())
                {
                    user = db.WEBUSERS.FirstOrDefault(u => u.EMAIL == model.Name &&
u.PASSWD == model.Password);

                }
                // если нашли пользователя с введённым логином и паролем, то
                // запоминаем его и делаем переадресацию на стартовую страницу
                if (user != null)
                {
                    FormsAuthentication.SetAuthCookie(model.Name, true);
                    return RedirectToAction("Index", "Invoice");
                }
                else
                {
```

```
                    ModelState.AddModelError("", "Пользователя с таким логином и паролем
не существует");
                }
            }

            return View(model);
        }

        [Authorize(Roles = "admin")]
        public ActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                WEBUSER user = null;
                using (DbModel db = new DbModel())
                {
                    user = db.WEBUSERS.FirstOrDefault(u => u.EMAIL == model.Name);
                }
                if (user == null)
                {
                    // создаём нового пользователя
                    using (DbModel db = new DbModel())
                    {

                        // получаем новый идентификатор с помощью генератора
                        int userId = db.NextValueFor("SEQ_WEBUSER");
                        db.WEBUSERS.Add(new WEBUSER {
                            WEBUSER_ID = userId,
                            EMAIL = model.Name,
                            PASSWD = model.Password
                        });
                        db.SaveChanges();

                        user = db.WEBUSERS.Where(u => u.WEBUSER_ID ==
userId).FirstOrDefault();

                        // находим роль manager
                        // Эта роль будет ролью по умолчанию, т.е.
                        // будет выдана автоматически при регистрации
                        var defaultRole = db.WEBROLES.Where(r => r.NAME ==
"manager").FirstOrDefault();

                        // назначаем вновь добавленному пользователю роль по умолчанию
                        if (user != null && defaultRole != null)
                        {
                            db.WEBUSERINROLES.Add(new WEBUSERINROLE
                            {
                                WEBUSER_ID = user.WEBUSER_ID,
                                WEBROLE_ID = defaultRole.WEBROLE_ID
                            });
                            db.SaveChanges();
                        }
                    }
                    // если пользователь удачно добавлен в бд
                    if (user != null)
                    {
                        FormsAuthentication.SetAuthCookie(model.Name, true);
                        return RedirectToAction("Login", "Account");
```

```
                }
            }
            else
            {
                ModelState.AddModelError("", "Пользователь с таким логином уже
существует");
            }
        }

        return View(model);
    }

    public ActionResult Logoff()
    {
        FormsAuthentication.SignOut();
        return RedirectToAction("Login", "Account");
    }
  }
}
```

Обратите внимание на атрибут `[Authorize(Roles = "admin")]`. Он обозначает, что действие по регистрации пользователей может производить только пользователь с ролью admin. Этот механизм называется фильтрами авторизации. Он нём будет сказано чуть позже.

При регистрации мы добавляем нового пользователя в БД, а при логине просто смотрим, есть ли такой пользователь. И если пользователь найден, то с помощью аутентификации форм устанавливаем куки

```
FormsAuthentication.SetAuthCookie(model.Name, true);
```

Вся информация о пользователе в Asp.Net MVC хранится в свойстве HttpContext.User, которое представляет реализацию интерфейса **IPrincipal**, который определён в пространстве имён System.Security.Principal.

Интерфейс **IPrincipal** определяет свойство **Identity**, которое хранит объект интерфейса **IIdentity**, который описывает текущего пользователя.

Интерфейс **IIdentity** содержит следующие свойства:

- **AuthenticationType**: тип аутентификации
- **IsAuthenticated**: если пользователь аутентифицирован, то возвращает true
- **Name**: имя пользователя в системе

Для определения аутентифицирован ли пользователь, ASP.NET MVC принимает от браузера куки, и если пользователь аутентифицирован, у свойства IIdentity.IsAuthenticated устанавливается значение true, а в свойство Name получает в качестве значения имя пользователя.

Теперь добавим элементы авторизации. Для этого воспользуемся механизмом универсальных провайдеров.

Универсальные провайдеры предоставляют уже готовый функционал авторизации. Но в то же время эти провайдеры обладают достаточной гибкостью – в частности мы можем их переопределить по своему усмотрению. При этом нам

необязательно переопределять и использовать все четыре провайдера. Что довольно удобно, особенно в ситуации, когда нам не нужны все навороты ASP.NET Identity, а требуется построить очень простенькую систему авторизации.

Итак, переопределим провайдер ролей. Для этого добавим через NuGet пакет Microsoft.AspNet.Providers.

Теперь определим сам провайдер ролей. Для этого сначала добавим в проект папку *Providers* и затем в него добавим новый класс MyRoleProvider:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using FBMVCExample.Models;

namespace FBMVCExample.Providers
{
    public class MyRoleProvider : RoleProvider
    {
        /// <summary>
        /// Возвращает список имён ролей у пользователя
        /// </summary>
        /// <param name="username">Имя пользователя</param>
        /// <returns></returns>
        public override string[] GetRolesForUser(string username)
        {
            string[] roles = new string[] { };
            using (DbModel db = new DbModel())
            {
                // Получаем пользователя
                WEBUSER user = db.WEBUSERS.FirstOrDefault(u => u.EMAIL == username);
                if (user != null)
                {
                    // заполняем массив доступных ролей
                    int i = 0;
                    roles = new string[user.WEBUSERINROLES.Count];
                    foreach (var rolesInUser in user.WEBUSERINROLES)
                    {
                        roles[i] = rolesInUser.WEBROLE.NAME;
                        i++;
                    }
                }
            }
            return roles;
        }

        /// <summary>
        /// Создаение новой роли
        /// </summary>
        /// <param name="roleName">Имя роли</param>
        public override void CreateRole(string roleName)
        {

            using (DbModel db = new DbModel())
            {
                WEBROLE newRole = new WEBROLE() { NAME = roleName };
                db.WEBROLES.Add(newRole);
                db.SaveChanges();
            }
        }
```

```
        /// <summary>
        /// Возвращает присутствует ли роль у пользователя
        /// </summary>
        /// <param name="username">Имя пользователя</param>
        /// <param name="roleName">Имя роли</param>
        /// <returns></returns>
        public override bool IsUserInRole(string username, string roleName)
        {
            bool outputResult = false;
            using (DbModel db = new DbModel())
            {
                var userInRole =
                    from ur in db.WEBUSERINROLES
                    where ur.WEBUSER.EMAIL == username && ur.WEBROLE.NAME == roleName
                    select new { id = ur.ID };

                outputResult = userInRole.Count() > 0;
            }
            return outputResult;
        }

        public override void AddUsersToRoles(string[] usernames, string[] roleNames)
        {
            throw new NotImplementedException();
        }

        public override string ApplicationName
        {
            get { throw new NotImplementedException(); }
            set { throw new NotImplementedException(); }
        }

        public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
        {
            throw new NotImplementedException();
        }

        public override string[] FindUsersInRole(string roleName, string usernameToMatch)
        {
            throw new NotImplementedException();
        }

        public override string[] GetAllRoles()
        {
            throw new NotImplementedException();
        }

        public override string[] GetUsersInRole(string roleName)
        {
            throw new NotImplementedException();
        }

        public override void RemoveUsersFromRoles(string[] usernames, string[] roleNames)
        {
            throw new NotImplementedException();
        }

        public override bool RoleExists(string roleName)
        {
            throw new NotImplementedException();
        }
    }
}
```

В целях демонстрации переопределено три метода. Первый из них - GetRolesForUser позволяет получать набор ролей для определённого пользователя. Второй метод - CreateRole - предполагает создание роли. И третий метод - IsUserInRole - определяет, выполняет ли пользователь определённую роль в системе.

Чтобы использовать провайдер ролей в приложении, надо добавить его определение в файл конфигурации. Откроем файл web.config и удалим из него определение провайдеров, которые были добавлены автоматически при добавлении пакета Microsoft.AspNet.Providers. И добавим туда вместо этого в пределах узла system.web добавим наш провайдер:

```
  <system.web>
    <authentication mode="Forms">
      <forms name="cookies" timeout="2880" loginUrl="~/Account/Login"
defaultUrl="~/Invoice/Index"/>
    </authentication>
    <roleManager enabled="true" defaultProvider="MyRoleProvider">
      <providers>
        <add name="MyRoleProvider" type="FBMVCExample.Providers.MyRoleProvider" />
      </providers>
    </roleManager>
  </system.web>
```

И теперь мы можем разграничить доступ к методам различных контроллеров с помощью атрибута *Authorize*. Мы уже видели его применение в контроллере `AccountController`:

```
        [Authorize(Roles = "admin")]
        public ActionResult Register()
        {
....
```

Данный фильтр можно применять как на уровне контроллера в целом, так и для отдельного действия контроллера. Давайте добавим разграничение прав для наших трёх основных контроллеров CustomerController, InvoiceController и ProductController. В нашем случае пользователь с ролью manager может смотреть и править данные во всех трёх таблицах. Установка фильтра для контроллера InvoiceController будет выглядеть следующим образом:

```
    [Authorize(Roles = "manager")]
    public class InvoiceController : Controller
    {
        private DbModel db = new DbModel();

        // Отображение представления
        public ActionResult Index()
        {
            return View();
        }
…
```

Для остальных контроллеров установка фильтра происходит аналогично.

Ну, вот и всё. Надеюсь, эта статья помогла вам разобраться в особенностях написания веб приложения на C# с использованием Entity Framework и ASP.NET MVC под СУБД Firebird.

Ссылки

Исходные коды примера приложения
Готовая БД 2.5 и 3.0

www.ibase.ru, www.ibsurgeon.com

support@ibase.ru, support@ib-aid.com