**Лабораторная работа 5. Разработка модели**

***Упражнение 1. Создание приложения с реализацией хранения данных***

**Создание проекта**

В этом упражнении вы разработаете новое приложение, которое будет выводить на главной странице из базы данных информацию о кредитах, также будет реализована возможность пользователю подать заявку на получение кредита, и эта заявка сохранится в базе данных.

1. Создайте новый проект ASP.NET Web Application с именем **MvcCreditApp1**.

2. В окне выбора шаблона в разделе **Шаблоны ASP.NET** укажите **MVC**, в перечне флажков добавления папок и ссылок также выберите **MVC** – будет создан проект со структурой папок, необходимых для создания MVC приложений.

3. Изучите структуру проекта, раскройте папку **Content**, содержащую используемые стили, откройте контроллер HomeController, реализующий три метода действия и откройте советующие эти метолам представления: About, Contact и Index, изучите их содержимое.

4. Запустите проект и протестируйте работу готового шаблона сайта.

// Выполнено

**Реализация модели**

Данные, с которыми будет работать клиент, должны быть представлены моделями. Для данной задачи можно выделить две области данных: информация о кредите и информация о заявке на кредит, соответственно, необходимо создать две модели.

1. Добавьте в папку **Models** класс Credit, который будет реализовывать модель данных о кредите.

2. Добавьте в него код, описывающий модель кредита:

public class Credit

{

// ID кредита

public virtual int CreditId { get; set; }

// Название

public virtual string Head { get; set; }

// Период, на который выдается кредит

public virtual int Period { get; set; }

// Максимальная сумма кредита

public virtual int Sum { get; set; }

// Процентная ставка

public virtual int Procent { get; set; }

}

3. Добавьте в папку **Models** класс Bid, который будет реализовывать модель данных о заявке на кредит.

4. Добавьте в него код, описывающий модель заявки на кредит:

public class Bid

{

// ID заявки

public virtual int BidId { get; set; }

// Имя заявителя

public virtual string Name { get; set; }

// Название кредита

public virtual string CreditHead { get; set; }

// Дата подачи заявки

public virtual DateTime bidDate { get; set; }

}

**Применение Entity Framework**

Для доступа к данным будем использовать **Entity Framework**. Этот фреймворк позволяет абстрагироваться от структуры конкретной базы данных и вести все операции с данными через модель.

**Entity Framework** в сочетании с **LINQ** (Language-Integrated Query) представляет собой реализацию ORM (объектно-реляционное отображение (object-relational mapping — ORM) для платформы .NET Framework от компании Microsoft. **Entity Framework** содержит механизмы создания и работы с сущностями базы данных через объектно-ориентированный код на языке C#.

В этом упражнении рассматривается подход **Code-First**, при использовании которого сначала определяется модель в коде, а затем, на ее основе создается база данных. Вы создадите две таблицы, описывающие данные клиента и его кредит Отношение между этими таблицами будет “один к одному” (one-to- one).

**Entity Framework** при работе с **Code First** требует определения ключа элемента для создания первичного ключа в таблице в БД. По умолчанию при генерации БД EF в качестве первичных ключей будет рассматривать свойства с именами Id или [Имя\_класса]Id (т. е. CreditId и BidId).

Поскольку проект создан по шаблону **MVC**, то библиотека **Entity Framework** уже добавлена в проект.

// Она сама не добавилась. Подчеркивает Entity. Нужно добавлять вручную через References ссылку System.Data.Entity. И много чего в этой лабораторной нужно добавлять вручную.

**Создание контекста данных**

1. Создайте контекст данных (он нужен для облегчения доступа к БД на основе модели), для этого добавьте в папку **Models** класс CreditContext, унаследованный от класса DbContext**.**

2. Добавьте ссылку на пространство имен System.Data.Entity в файле CreditContext.cs:

using System.Data.Entity;

3. В классе CreditContext объявите соответствующие таблицы (по традиции во множественном числе) базы данных с помощью свойств с типом DbSet:

namespace MvcCreditApp1.Models

{

public class CreditContext: DbContext

{

public DbSet<Credit> Credits { get; set; }

public DbSet<Bid> Bids { get; set; }

}

}

Этот класс контекста представляет полный слой данных, который можно использовать в приложениях. Благодаря **DbContext**, можно запросить, изменить, удалить или вставить значения в базу данных.

4. Постройте приложение.

// выполнено

**Создание базы данных на основе модели**

В этой части упражнения вы создадите базу данных и заполните ее значениями.

1. В папку **Models** добавьте класс CreditsDbInitializer унаследованный от класса DropCreateDatabaseIfModelChanges (при инициализации в случае изменения модели старая база будет удаляться и создаваться новая с начальными значениями), подключите требуемое пространство имен:

using System.Data.Entity;

...

public class CreditsDbInitializer : DropCreateDatabaseIfModelChanges<CreditContext>

{

}

2. Переопределите метод Seed() в котором создайте, например, три кредита и добавьте их в таблицу **Credits** с помощью метода Add() свойства Credits:

public class CreditsDbInitializer : DropCreateDatabaseIfModelChanges<CreditContext>

{

protected override void Seed(CreditContext context)

{

context.Credits.Add(new Credit { Head = "Ипотечный кредит", Period = 10, Sum = 1000000, Procent = 15 });

context.Credits.Add(new Credit { Head = "Образовательный кредит", Period = 7, Sum = 300000, Procent = 10 });

context.Credits.Add(new Credit { Head = "Потребительский кредит", Period = 5, Sum = 500000, Procent = 19 });

base.Seed(context);

}

}

3. Для генерации базы данных необходимо чтобы при запуске приложения создавался экземпляр класса **CreditsDbInitializer**. Для этого откройте файл **Global.asax** и добавьте в метод Application\_Start, который выполняется при старте приложения, создание объекта:

Database.SetInitializer(new CreditsDbInitializer());

4. Импортируйте в файл **Global.asax** пространства имен MvcCreditApp1.Models и System.Data.Entity:

using MvcCreditApp1.Models;

using System.Data.Entity;

***Упражнение 2. Настройка работы с данными***

В этом упражнении вы внесете изменения в контроллеры и представления для работы с данными.

**Настройка контроллера**

1. Откройте находящийся в папке **Controllers** контроллер HomeController.

2. Импортируйте пространство имен MvcCreditApp1.Models.

3. В поле класса контроллера создайте экземпляр контекста данных:

private CreditContext db = new CreditContext();

4. В методе Index() обратитесь к контексту и получите все записи о кредитах:

var allCredits = db.Credits.ToList<Credit>();

5. Далее создайте свойство Credits в объекте ViewBag и присвойте ему извлеченный список. Объект ViewBag является таким объектом, который передается в представление:

ViewBag.Credits = allCredits;

6. В итоге класс контроллера с измененным методом выглядит следующим образом:

using MvcCreditApp1.Models;

...

public class HomeController : Controller

{

private CreditContext db = new CreditContext();

public ActionResult Index()

{

var allCredits = db.Credits.ToList<Credit>();

ViewBag.Credits = allCredits;

return View();

}

}

Обратите внимание, что так как модели находятся в другом пространстве имен, хотя и в одном проекте, то его необходимо импортировать.

7. Постройте проект.

**Изменение представления для отображения данных**

1. Откройте находящиеся в папке **Home** представление Index.

2. Изучите разметку для позиционирования элементов, для отображения данных будет использован тот же принцип – в виде сетки с указанием строк и столбцов.

3. Замените сгенерированный код разметки на следующий:

@{

ViewBag.Title = "Главная";

}

<div class="row">

<div class="col-md-4">

<h3>Программы кредитования</h3>

<table>

<tr><td><p>Тип кредита</p></td><td><p>Период кредитования</p></td><td><p>Максимальная сумма</p></td><td><p>Ставка %</p></td></tr>

@foreach (var c in ViewBag.Credits)

{

<tr><td><p>@c.Head</p></td><td><p>@c.Period</p></td><td><p>@c.Sum</p></td><td><p>@c.Procent %</p></td></tr>

}

</table>

<p><a class="btn btn-default" href="/Home/CreateBid">Подать заявку на получение кредита &raquo;</a></p>

</div>

</div>

В этом коде создается таблица, в которой будут располагаться данные о программах кредитования. Вся информация помещается в ячейку сетки.

Конструкция @foreach (var c in ViewBag.Credits) использует синтаксис движка представления Razor (после символа @ согласно синтаксису, можно использовать выражения кода на языке C#).

В цикле реализован проход по элементам в объекте ViewBag.Credits, который был создан в методе контроллера, получены свойства каждого элемента и помещены в таблицу.

Ссылка href="/Home/CreateBid">Подать заявку на получение кредита означает адрес, по которому будет размещаться форма заявки на кредит.

4. Постройте и запустите приложение. После загрузки и генерации базы данных должна отобразится таблица с ранее определенными данными (см. рис.5.1).



*Рис. 5.1 Отображение таблицы данных*

// Получилось

**Реализация подачи заявки**

В этой части упражнения будет создан метод CreateBid, который будет отвечать за обработку ввода пользователя при подаче заявки.

1. Добавьте в контроллер **HomeController** обычный метод для получения информации о существующих кредитах и скопируйте в него код из метода Index():

private void GiveCredits()

{

var allCredits = db.Credits.ToList<Credit>();

ViewBag.Credits = allCredits;

}

2. В методе Index()замените соответствующий код на вызов метода.

// Комментим в индексе – переносим в гив кредитс.

3. Добавьте в контроллер **HomeController** два метода, определяющие одно действие **CreateBid**, в первом случае оно выполняется при получении запроса GET, а во втором случае – при получении запроса POST:

a. Первый метод ActionResult CreateBid() возвращает соответствующее представление c получением всех записей о кредитах и заявках:

[HttpGet]

public ActionResult CreateBid()

{

GiveCredits();

var allBids = db.Bids.ToList<Bid>();

ViewBag.Bids = allBids;

return View();

}

b. Второй метод принимает переданную ему в запросе POST модель newBid и добавляет ее в базу данных. В конце возвращается строка уведомительного сообщения:

[HttpPost]

public string CreateBid(Bid newBid)

{

newBid.bidDate = DateTime.Now;

// Добавляем новую заявку в БД

db.Bids.Add(newBid);

// Сохраняем в БД все изменения

db.SaveChanges();

return "Спасибо, <b>" + newBid.Name + "</b>, за выбор нашего банка. Ваша заявка будет рассмотрена в течении 10 дней.";

}

4. Добавьте представление **CreateBid**. // оно же Viev // Для этого нажмите на метод public ActionResult CreateBid() правой кнопкой и добавьте в проект новое представление, имя (CreateBid**)** и остальные параметры оставьте по умолчанию, проверьте, что флажок *Использовать страницу макета* включен, но само поле пустое.

5. Добавьте в конце файла следующий код отображения существующих заявок и создания формы ввода данных:

<h3>Существующие заявки</h3>

<table>

<tr><td><p>Имя</p></td><td><p>Тип кредита </p></td></tr>

@foreach (var c in ViewBag.Bids)

{

<tr><td><p>@c.Name</p></td><td><p>@c.CreditHead </p></td></tr>

}

</table>

<h3>Форма подачи заявки по программам кредитования</h3>

<form method="post" action="">

<table>

<tr><td><p>Введите свое имя </p></td><td><input type="text" name="Name" /> </td></tr>

<tr><td><p>Выберите из списка тип кредита :</p></td>

<td>

<select name="CreditHead">

@foreach (var cr in ViewBag.Credits)

{

<option>@cr.Head</option>

}

</select>

</td>

</tr>

<tr><td><input type="submit" value="Отправить" /> </td><td></td></tr>

</table>

</form>

Обратите внимание на реализацию списка кредита с помощью цикла foreach, в дальнейшем будет реализована возможность добавления нового типа кредита и в этом случае в списке новый кредит отобразится.

// также можно делать итоговую

6. Постройте и запустите приложение. Введите данные о заявке и нажмите кнопку "Отправить". После этого заявка попадет в базу данных, а в браузере отобразится соответствующее уведомление.

// Результат

Спасибо, Славка Тислер, за выбор нашего банка. Ваша заявка будет рассмотрена в течении 10 дней.

7. Вернитесь на страницу **GreateBid** и обновите ее. Проверьте, что информация о существующих заявках отобразилась на странице.

// Получилось

Заполняется столбец заявки всеми персонажами

**Стилизация приложения**

В этом приложении по умолчанию применяется фреймворк Bootstrap. Для того, чтобы созданная вами таблица была оформлена в стиле Bootstrap, ей нужно назначить класс table: <table class = "table">.

1. Назначьте класс table таблицам на странице **GreateBid**.

2. Откройте находящийся в папке **Content** файл Bootstrap.css и добавьте в раздел table новый стиль (выделен жирным шрифтом):

table {

border-collapse: collapse;

border-spacing: 0;

border:6px solid silver;

vertical-align:middle;

text-align:center;

}

3. Для таблицы на странице **Index** добавьте дополнительный класс Bootstrap table-bordered к базовому классу **table**: <table class="table-bordered">, в этом случае добавятся границы для всех ячеек таблицы.

4. В файле \_Layout.cshtml добавьте ссылку на страницу с заявкой:

<ul class="nav navbar-nav">

...

<li>@Html.ActionLink("Послать заявку", "CreateBid", "Home")</li>

</ul>

5. Запустите приложение и убедитесь, что теперь к сайту применена стилизация (см. рис. 5.2, 5.3).



*Рис. 5.2 Применение стилизации к стартовой странице*



*Рис. 5.3 Применение стилизации к странице подачи заявки*

// Выполнено.